PEEDI

Powerful Embedded Ethernet Debug Interface

User Manual

Version 1.37

RONETIX
embedded development tools

April, 2010
Ronetix has made every attempt to ensure that the information in this document is accurate and complete. However, Ronetix assumes no responsibility for any errors, omissions, or for any consequences resulting from the use of the information included herein or the equipment it accompanies. Ronetix reserves the right to make changes in its products and specifications at any time without notice.
Any software described in this document is furnished under a license or non-disclosure agreement. It is against the law to copy this software on magnetic tape, disk, or other medium for any purpose other than the licensee's personal use.

Ronetix Development Tools GmbH
Waidhausenstrasse 13/5
1140 Vienna
Austria
Tel: +43-720-500315
+43-1 956 3138
Fax: +43-1- 8174 955 3464
Internet: www.ronetix.at
E-Mail info@ronetix.at

Acknowledgments:

ARM, ARM7, ARM9, ARM11, Cortex-M3, Cortex-A8 and Thumb are trademarks of ARM Ltd.
PowerPC and ColdFire are trademarks of Freescale Ltd.
Blackfin is trademark of Analog Devices Ltd.
Windows, Win32, Windows CE are trademarks of Microsoft Corporation.
Ethernet is a trademark of XEROX.
MIPS32 is a trademark of MIPS Technologies
AVR32 is a trademark of Atmel
All other trademarks are trademarks of their respective companies.

© 2005-2010 RONETIX GmbH
All rights reserved.

Change log

April 2010	- Added MPC8500 support - Added PPC405 support - Added PXA320 NAND FLASH support
December 2009	- Added **target** command - Added STM32 agent FLASH programming - Added Freescale iMX51 NAND FLASH support - Added BAD_BLOCK_TABLE NAND FLASH CFG parameter - Added SWAP_BI NAND FLASH CFG parameter
September 2009	- Added Marvell Feroceon support
August 2009	- Added NEXT_THREAD parameter in the OS section - Added option to use file path in the BASE parameter of OS section
July 2009	- Added Cortex-M3 SWO Stimulus Trace Ports support - Added target OS tasks awareness - Corrected Debugging Linux Kernel chapter
June 2009	- Added MPC5200 debug and FLASH support - Added AVR32 debug and FLASH support - Added 32-bit DCC transfer for ARM targets - Added STR9 FLASH secure/unsecure support
April 2009	- Added **reset detect** command - Added LPC29XX FLASH support - Added BASE_ADDR parameter in ST STM32 FLASH section
March 2009	- Added MIPS32 support - Added Cortex-M3 SWD support - Added PIC32 FLASH support
December 2008	- Added Cortex-A8 support
October 2008	- Added Freescale iMX27 NAND FLASH support - Added OneNAND FLASH support - Added Luminary Micro FLASH support - Added Freescale MPC8300 RCW explanation
September 2008	- Added COREn_DATASPACE parameter - Replaced COREn_WORKSPACE_ADDR and COREn_WORKSPACE_LEN parameters with single COREn_WORKSPACE parameter
August 2008	- Added unique serial number support in FLASH programmer
July 2008	- Added Cortex-M3 support - Added ST STM32 FLASH support - Added **flash this option** ST STM32 FLASH command - Added new GDB registers frame support for MPC5500 targets
June 2008	- Added **flash this secure** AT91SAM7 FLASH command - Added **flash read** command description - Licenses now can be loaded from an external file - Fixed DCC example code - DBGREQ_OUTPUT is now available for all JTAG targets
May 2008	- Added Freescale PowerQUICC II Pro MPC83XX support - Added **beep** command - Added COREn_MEMMAP config parameter for Blackfin - Added support of 2048 bytes of TFTP transfer packet size
March 2008	- Added **flash this hidden** generic FLASH command - Added **flash this nvmbit** AT91SAM7 FLASH command - Added WDKICK_TIME in ARM [TARGET] section - Complemented **break add** command
February 2008	- Added note about TMS470's AWD in CPU specific considerations - Added **echo** command
January 2008	- Added ARM11 support - Added Blackfin NFC programming support - Added Blackfin SPI FLASH programming support - Added ALE_ASSERT/RELEASE, CLE_ASSERT/RELEASE NAND config parameters - Added Freescale MCF5206 BDM support

	- Reworked **reset** command to accept target reset options
November 2007	- Added Analog Devices Blackfin DataFlash programming support
September 2007	- Added JTAG_TDO_DELAY CFG parameter - Added Analog Devices Blackfin NAND FLASH programming support - Added Atmel AT91SAM9263 DataFlash programming support - Added Freescale MCF5272 BDM support
June 2007	- Added Analog Devices Blackfin debug support - Added Analog Devices Blackfin FLASH programming support - Added **memory and/or** commands - Added **flash multi verify** and **flash multi blank** commands
March 2007	- Added Freescale BDM ColdFire debug support - Added Freescale BDM ColdFire FLASH programming support - Added Command Line Interface on the RS232 line - Added second possible argument of RESET_TIME in ARM [TARGET] section - Added DBGREQ_OUTPUT in ARM [TARGET] section - Added NAND FLASH YAFFS support - Added JFFS2_NO_EM NAND FLASH CFG argument - Added **info ctrl** command - Added **clock** command - Added **jtag** command
November 2006	- Added Freescale Nexus enabled MPC5500 debug support - Added Freescale MPC5500 FLASH programming support - Added NAND FLASH JFFS2 support - Added NAND FLASH support on XScale CPUs - Added WAKEUP_TIME in XScale [TARGET] section - Added **info spr** command - Added **memory read64** and **memory write64** commands - Added **execute** command
September 2006	- Complemented STR7 family specifics - Added AUTO_LOCK parameter to the STR7 family [FLASH] section - Added **memory test** command
August 2006	- Added NAND FLASH support - Added ALLOW_ZERO_KEYS in TMS470 family [FLASH] section - Added support of using expression instead of a value in the commands - Added **clock** command - Replaced sample configuration files with a link for download
July 2006	- Added COREn_PATH configuration parameter in [TARGET] section - Added COREn_FILE configuration parameter in [TARGET] section - FILE_XXXX parameters replaced by single FILE parameter in [FLASH] sections - Changed File Path Convention - Added **memory verify** command - Added **flash dump** command - Added execution of the scripts defined in target configuration file from CLI - Added source code of example target side DCC functions
June 2006	- Added TI TMS470 series microcontroller support - Added Atmel DataFlash support - Added ucLinux debugging support - Added DCC over TCP support - Added **memory multi load** command - Improved XScale exception vectors handling
April 2006	- Added ST STR7 series microcontroller support - Added FLASH auto detect
March 2006	- Added Intel XScale CPU support - Added Freescale MAC7100 series microcontroller support - Reworked target connect procedure, see Boot procedure - Removed STARTUP_TIME configuration parameter - Added WAKEUP_TIME configuration parameter
February 2006	- The sample target configuration file spitted to three separated files - Complemented Philips LPC2000 family specifics - Removed boot script explanation in RedBoot configuration
January 2006	- Added **flash multi program** command

	- Added **flash multi erase** command - Complemented Multiple core support
December 2005	- Added Philips LPC2000 series microcontroller support - Added chapter “CPU specific considerations” - Added Appendix A: Sample Target Configuration Files - Updated chapter FAQ
November 2005	- Added Angel Debug Protocol usage explanation - The chapter “Software Installation” moved to the “Software Installation Guide”
September 2005	- Updated Boot sequence - Added multiple core explanation - Added multiple flash explanation - Added **flash set** command - Added **flash lock** command - Added **flash unlock** command - Added **flash query** command - Removed **flash write** command - Added Atmel AT91SAM7 series microcontroller support
August 2005	- Added parameter STARTUP_MODE in section PLATFORM_ARM - Added Boot sequence - Complemented firmware update procedure
July 2005	- First release

1 INTRODUCTION 10

1.1 PEEDI in the development process 10
1.1.1 Single developer environment 10
1.1.2 Multiple developers environment 11

1.2 PEEDI in the manufacturing process 12
1.2.1 PEEDI as a standalone FLASH programmer 12
1.2.2 PEEDI as a device tester 13
1.2.3 High productivity with the Multi Core feature 13

2 INSTALLATION 14

2.1 Hardware installation 14
2.1.1 Connection instructions 15

2.2 Software installation 16

3 USING PEEDI 17

3.1 PEEDI interface 17

3.2 Setup with RedBoot 18
3.2.1 RedBoot Configuration 18
3.2.2 Firmware update procedure 19
3.2.2.1 Update via RS232 19
3.2.2.2 Update via Ethernet 19
3.2.3 RedBoot commands used with PEEDI 20
fconfig 20
update 20
memtest 20

3.3 Configure PEEDI 21
3.3.1 Network configuration 21
3.3.2 Target configuration file 21
Section LICENSE 21
Section DEBUGGER_PROTOCOL 21
Section TARGET 22
Section PLATFORM_ARM 22
Section PLATFORM_ARM11 26
Section PLATFORM_CortexM3 & Section PLATFORM_CortexM3_SWD 26
Section PLATFORM_CortexA8 26
Section PLATFORM_XSCALE 27
Section PLATFORM_MPC5500 27
Section PLATFORM_MPC8300 28
Section PLATFORM_MPC5200 29
Section PLATFORM_MPC8500 29
Section PLATFORM_PPC400 30
Section PLATFORM_COLDFIRE 30
Section PLATFORM_BLACKFIN 30
Section PLATFORM_MIPS32 32
Section PLATFORM_AVR32 32
Section INIT 32
Section FLASH 33
Section OS 44
Section SERIAL 46
Section TELNET 46
Section DISPLAY 47
Section ACTIONS 47

3.4 CPU specific considerations 49
3.4.1 Atmel AT91SAM7 family 49
3.4.2 Atmel AVR32UC family 49
3.4.3 Philips LPC2000 family 49
3.4.4 Freescale MAC7100 family 49
3.4.5 ST STR7 family 50
3.4.6 ST STR9 family 50
3.4.7 ST STM32 family 51
3.4.8 TI TMS470 family 51
3.4.9 Intel XScale family 52
3.4.10 NAND FLASH 52
3.4.11 Freescale MPC5500 family 53
3.4.12 Freescale PowerQUICC II Pro MPC83XX family 55
3.4.13 Freescale MCF5200 family 55
3.4.14 Analog Devices Blackfin family 55

3.5 Boot sequence 56

3.6 Multiple core support 58

3.7 Script execution using the front panel interface 59

3.8 Serial Interface 60

3.9 ARM DCC Interface 60

3.10 Working with Insight/gdb 61

3.11 Working with other debuggers 63
3.11.1 ARM Developer Suite (ADS) v1.2, AXD debugger 63
3.11.2 ARM RealView Developer Suite (RVDS) v2.2, RealView debugger 66
3.11.3 Green Hills MULTI v3.6 for ARM 70
3.11.4 IAR Embedded Workbench for ARM v4.30 72

3.12 Debugging Linux kernel 74

3.13 Target OS tasks awareness and debugging uClinux applications 74

3.14 Working with CLI (Command Line Interface) 76
3.14.1 File path convention 76
3.14.2 CLI commands 78
help 78
transfer 78
type 78
wait 78
core 79
clock 79
adp 79
run 79
go 80
step 80
execute 80
set 80
halt 81
reset 81
reboot 81
echo 82
jtag 82
beep 82
jtag 82
target 82

quit 82
info 83
info flash 83
info registers 83
info target 83
info config 83
info ice 84
info cp15, info cp14 84
info spr 85
info ctrl 85
info breakpoint 85
memory 85
memory read 85
memory write 86
memory or 86
memory and 87
memory crc 87
memory load 87
memory multi load 88
memory verify 88
memory dump 89
memory test 89
flash 89
flash set 89
flash blank 90
flash erase 90
flash lock 90
flash unlock 90
flash query 91
flash program 91
flash multi erase 91
flash multi blank 92
flash multi program 92
flash multi verify 92
flash verify 93
flash dump 93
flash read 93
flash info 94
flash find 94
flash this 94
flash this hidden 94
flash this nvmbit 95
flash this secure 95
flash this option 95
breakpoint 95
breakpoint add 95
breakpoint add hard 96
breakpoint add watch 96
breakpoint delete 96
breakpoint list 96
card 97
card cd 97
card md 97
card rd 97
card dir 97
card copy 98
card type 98
card delete 98
card rename 98
eeprom 98
eeprom dir 98
eeprom copy 99

eeprom type 99
eeprom delete 99
eeprom rename 99
eeprom format 99
eeprom alias 100
3.14.3 Using aliases 100
3.14.4 Using scripts 101

3.15 Working with the FLASH programmer 101

3.16 Multiple FLASH support 102

3.17 Working with a MMC/SD memory card 102

3.18 JTAG cable adapters 103

3.19 PEEDI licenses 103

4 SPECIFICATIONS 105

5 FAQ 117

6 GLOSSARY 120

7 PEEDI PACKAGE CONTENTS 122

8 WARRANTY 123

APPENDIX A: SAMPLE TARGET CONFIGURATION FILES 124

1 Introduction

PEEDI (Powerful Embedded Ethernet Debug Interface) is an EmbeddedICE solution that enables you to debug software running on ARM processor cores via the JTAG port. JTAG is an IEEE standardized protocol that enables full control of the CPU core, giving the opportunity to debug embedded software, which extremely reduces Time-To-Market and increases the quality of the end product.

PEEDI is a debugging and development tool that provides the ability to see what is taking place in the target system and control its behavior. PEEDI provides the services needed to perform all debugging operations. It receives command packets over the communication link and translates them into JTAG operations that are needed to provide the specific service. It can control the operation of the target processor and target system, start and stop the processor's execution; it can set breakpoints in a program, examine and store values in the processor's registers, and examine and store program code or data in the target system's memory.

PEEDI can work in cooperation with a host computer or autonomously using MMC/SD card.

1.1 PEEDI in the development process

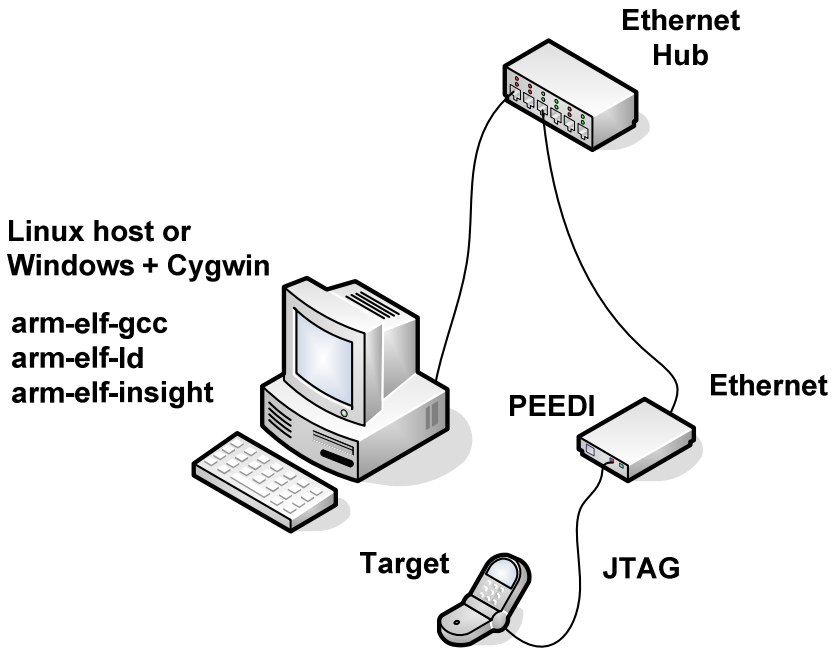
In the development process PEEDI can be used mainly as a debugger JTAG interface and FLASH programmer.

Two major configurations are possible here:

1.1.1 Single developer environment

Using the developer's PC as a host computer – this is suitable for small projects. Here all necessary tools for compiling and debugging the project must be installed on the developers PC, including file server (TFTP, FTP or HTTP) allowing PEEDI to retrieve configuration files or executable images. In this configuration the developer's PC must be connected to PEEDI in a common LAN (using crossover patch cable or via Ethernet hub/switch).

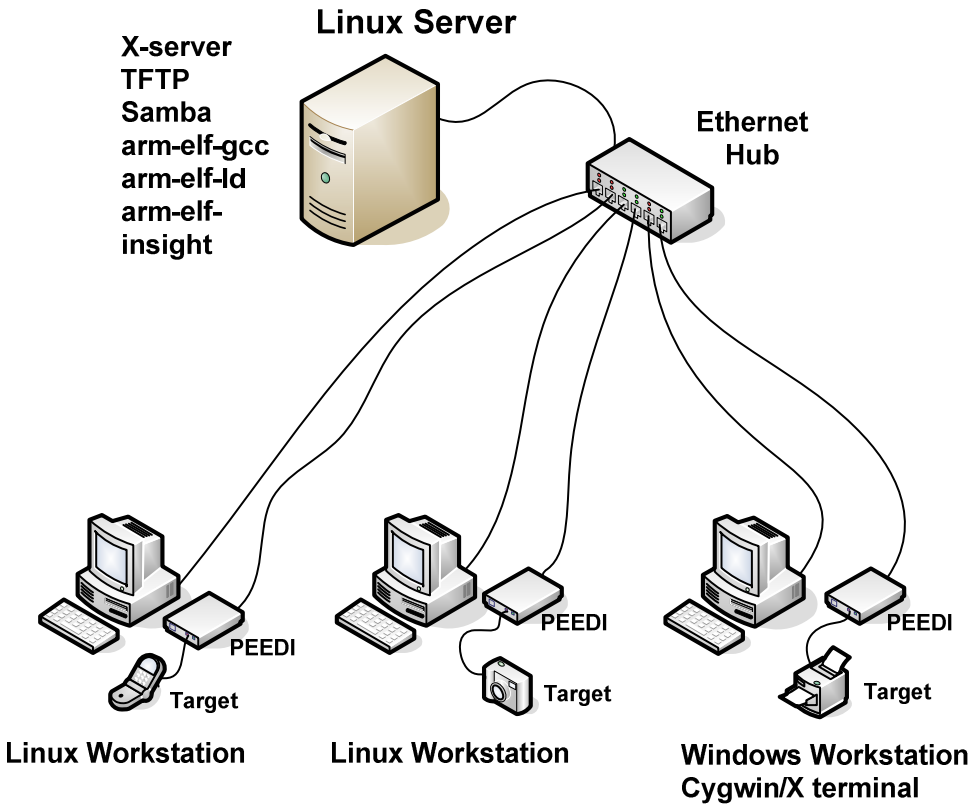
A simple configuration:



1.1.2 Multiple developers environment

Dedicated server with all the necessary development tools installed is used for a host. The developer uses his PC only as a graphical terminal to logon to the server and work there. This way no specific software is installed on the developer's PC, so it is very easy to set another working environment for a new developer for the project - just add a new user on the server and make a copy of the project source and make files in the user's home directory. Of course any source control tool, such as CVS or Visual Source Safe, can be used for synchronizing the project files. In this configuration all devices (the server, the developers' PCs and all PEEDIs) must be connected in a common LAN.

An example of multiple programmer development:



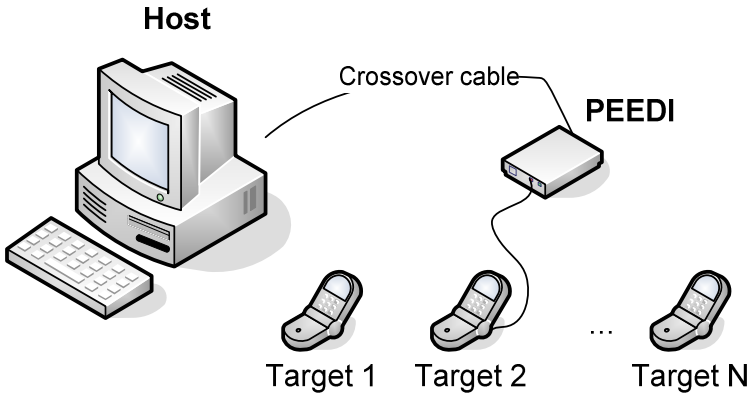
1.2 PEEDI in the manufacturing process

PEEDI can be used in the manufacture process as a tool for testing the device after it is assembled and as a FLASH programmer to program the device firmware. In both scenarios the host computer is not required because all the operations that must be performed can be formed as script files and executed using the PEEDI's front panel interface. If all the necessary files are stored on the MMC/SD card the Ethernet connection is not required as well.

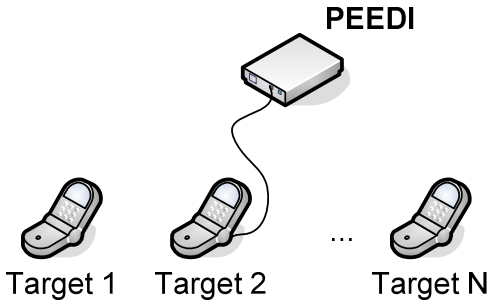
1.2.1 PEEDI as a standalone FLASH programmer

PEEDI can be used as a FLASH programmer in two ways:

- The first way is to connect to PEEDI via telnet and execute FLASH command and script files from the command line interface (CLI). This is the more sophisticated way; because you can see all the status messages in an easy understandable format i.e. warnings and errors.



- The second way is to use the front panel interface to choose, start and observe the status of scripts, which invokes the desired FLASH commands. Here you can define an AUTORUN script to be executed every time a target is connected; this way there is no need to start the script manually - very useful and time saving when great volumes of targets are to be programmed.



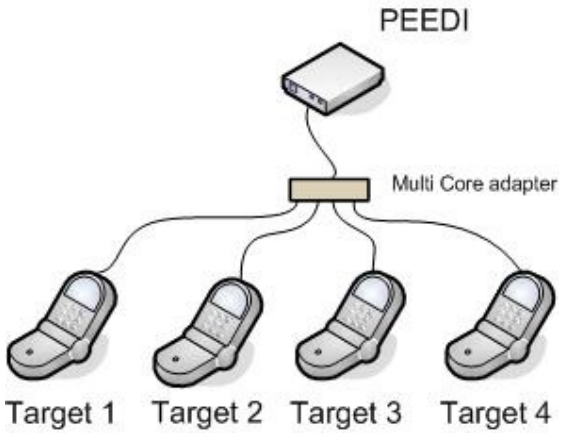
1.2.2 PEEDI as a device tester

Here the PEEDI can be used in the same manner as in the previous section – making telnet connection or through the front panel interface.
Depending on the specifics of what is to be tested two major ways can be followed:

- Execute commands that directly make some sort of test i.e. **flash verify**, **memory crc**, etc.
- Download executable code into target, which will perform the desired test and set a CPU register or memory on exit to a value showing the result of the test. Using this method, there is almost no test that could not be performed.

1.2.3 High productivity with the Multi Core feature

If you acquire the Multi Core feature you can quadruple you productivity because you can work on four targets simultaneously using only a singe PEEDI. The targets must be chained using the multiple core cable adapters available from Ronetix.



2 Installation

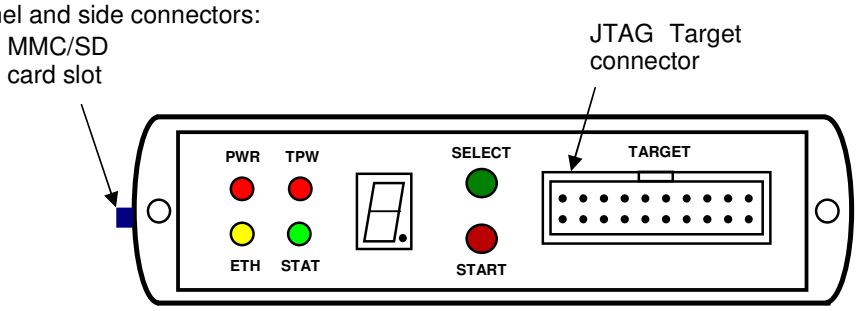
This chapter will explain how to connect PEEDI to the target and how to configure all the tools necessary for development.

Two major steps must be followed in order to set up a working PEEDI:

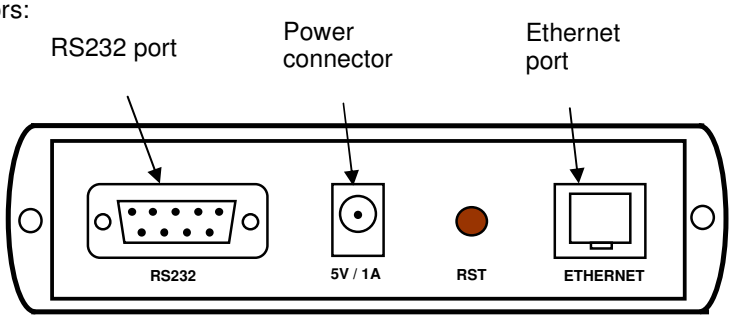
- Connect all required cables, this including power cord, target cable and if necessary the Ethernet cable, which will provide connection to a host computer or file servers. This is explained in the hardware installation chapter.
- Install and configure insight/gdb debugger. This is explained in the software installation chapter.

2.1 Hardware installation

Front panel and side connectors:



Rear panel connectors:

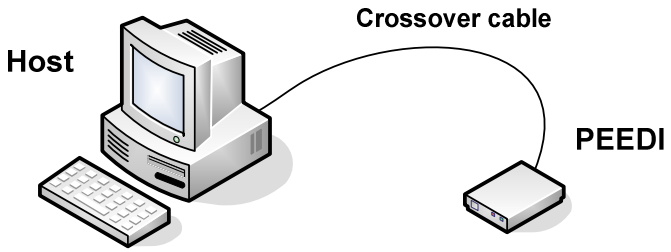


2.1.1 Connection instructions

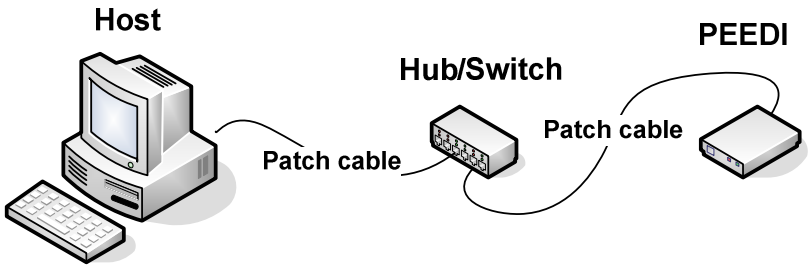
To connect the PEEDI interface unit to your host and to the target hardware:

1. Connect the host computer to an Ethernet network or directly to the PEEDI as required:

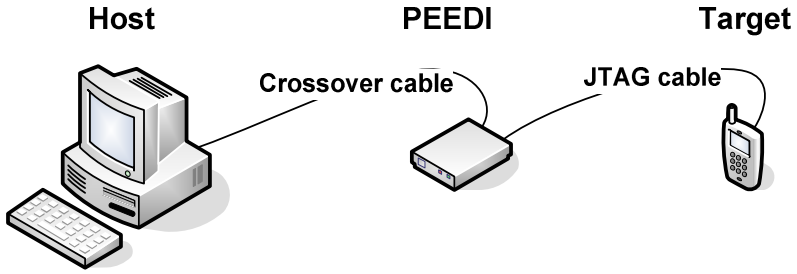
A. Direct host connection



B. LAN Connection



2. Connect the PEEDI interface unit to the target hardware, using the supplied JTAG adapter and cable. The JTAG adapter must be on the PEEDI side of the JTAG cable. If your target JTAG port pinout is not standard, you may need to make your own target cable considering the PEEDI JTAG connector pinout.



3. Power up the target hardware
4. Connect the external power supply to the PEEDI and to the power outlet.
5. When PEEDI boots, if you have a terminal connected to the RS232 port of PEEDI you will see some status messages.

2.2 Software installation

See “Cross development with GNU toolchain and Eclipse“:
http://www.ronetix.at/software.html

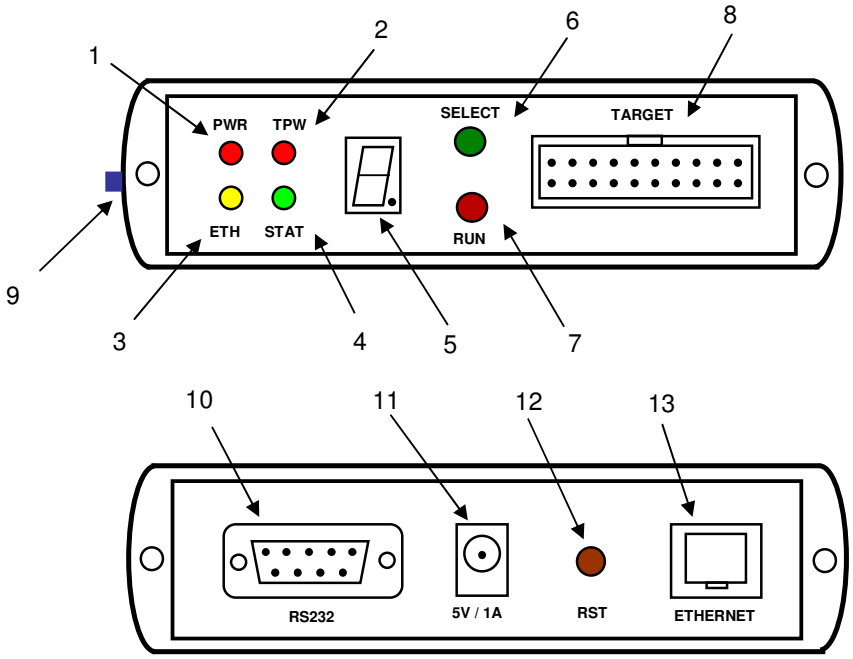
3 Using PEEDI

This chapter will explain PEEDI's operating modes, PEEDI's interface and the basic steps of configuring the software tools for working with PEEDI.

To start using PEEDI you need to:

- configure network settings
- make target configuration file

3.1 PEEDI interface



1	Power LED
2	Target power LED
3	Ethernet connect/activity LED
4	Target connect/activity LED
5	Script number/status LED display
6	Next script button
7	Start script button
8	Target connector
9	MMC/SD card slot
10	RS232 port
11	Power supply
12	Reset button
13	Ethernet port

3.2 Setup with RedBoot

RedBoot is a bootstrap loader, which during normal boot-up is used to load and launch PEEDI's executable image. RedBoot is also used to update PEEDI's firmware and to configure network settings, which are later used by PEEDI. RedBoot has some useful testing facilities like **ping** and **memtest**.

3.2.1 RedBoot Configuration

RedBoot and PEEDI share the same network settings. To set the network you need to connect a simple terminal application set to 115200, 8, N, 1 (for example HyperTerminal) to the PEEDI's RS232 port using a serial straight-through cable with DB9M (male) and DB9F (female) connectors on each end. Next step is to restart PEEDI by pressing the RESET button while holding both front panel buttons pressed. This will tell RedBoot not to load and launch the PEEDI executable if available, but to wait for connection on RS232 or Ethernet. While rebooting RedBoot should output some diagnostic information on the serial port which you should see. When RedBoot is ready to accept commands, it will show the command line prompt "RedBoot> ". Now you can use the **fconfig** command to set and save to FLASH all the parameters. When asked for different parameters please enter the following:

WARNING:
If PEEDI is set to get its network settings from a DHCP server and if the Ethernet cable is unplugged or there is no DHCP server on the Ethernet, it may take some time for PEEDI to boot. To avoid this, make sure PEEDI can reach a DHCP server.

```
Use DHCP for network configuration: yes /[no] [ENTER]
Gateway IP address: X.X.X.X
Local IP address: X.X.X.X
Local IP address mask: X.X.X.X
Default server IP address, used by RedBoot and PEEDI: X.X.X.X
```

Note:
*Instead of **X**'s enter IP address digits*

Next you will be asked for the path of the configuration file:

```
Target config file path:
```

Accepted paths for the different protocols are:

```
tftp://server/sub_directory/filename.cfg
ftp://user:password@server/sub_directory/filename.cfg
http://server/sub_directory/filename.cfg
card://sub_directory/filename.cfg
```

Note:
A server is indicated by its IP address.

Now you may enter DNS server used by RedBoot to resolve hostnames.

```
DNS server IP address:
```

If left blank and PEEDI is set to get the network configuration from DHCP server, the DNS server IP will also be taken from the DHCP.

Next you will be prompted for the RedBoot telnet port:

```
RedBoot telnet port: 23
```

Finally you may enter the update command default file path:

```
Update filepath:
  <1> - http://www.ronetix.at/download/firmware/fw_peedi_revA_last.bin
```

```
  <2> - tftp://192.168.3.1/fw_peedi_rev.A_last.bin
  <custom path>
Path: 1
```

If you have changed some of the parameters you will be ask to save them at the end. If you confirm to save them they will be take effect after the next start.

3.2.2 Firmware update procedure

First of all you need to reset PEEDI by pressing the RESET button on the back while holding both front panel buttons pressed. This will tell RedBoot not to load and launch the PEEDI executable, but to wait for connection on RS232 or Ethernet.
Entering the RedBoot command line prompt can be done using two different ways: via RS232 port using serial straight-through cable and a simple terminal application set to 115200, 8, N, 1 or if the network is configured you can connect using telnet application.
Once in the RedBoot's command line prompt (verify by pressing ENTER, RedBoot's prompt should appear – "RedBoot> "), you can update the firmware the following ways:

3.2.2.1 Update via RS232

If you want to update PEEDI via RS232 your terminal application must support XMODEM or YMODEM protocols. Now execute:

```
RedBoot> update xmodem
```
or
```
RedBoot> update ymodem
```

to tell RedBoot to start listening on RS232 port for incoming packets. Next tell your terminal application to start downloading the PEEDI firmware.

3.2.2.2 Update via Ethernet

Now you may use "update" command to update the PEEDI firmware. You can update using TFTP, HTTP. The syntax of the "update" command is:

```
update [FILEPATH]
```

The command shown below will attempt to download the firmware using the default filepath entered while configuring RedBoot using the **fconfig** command or the path used when last **update** command is invoked:

```
RedBoot> update
```

If not changed, the default update path points to the last version of the firmware directly on the RONETIX web site. If **update 1** is entered also the last version of the firmware directly from the RONETIX web site will be downloaded.

The following command will attempt to download the firmware using the HTTP protocol from a directory on the server (this syntax can be used with TFTP too):

```
RedBoot> update http://server/subdir/file.bin
```

After you enter the command using your specific conditions, if the host is accessible and the file is present you should see this:

```
RedBoot> update
load -r -m tftp -b 0x100000 -h 192.168.1.1 /fw_peedi_revA_last.bin
.....................................................................
Raw file loaded 0x00100000-0x002ab1bf, assumed entry at 0x00100000

Current Firmware:
-----------------
  Hardware Ver. : 1.2
  Software Ver. : 1.0
```

```
New Firmware:
-------------
  Hardware Ver. : 1.2
  Software Ver. : 1.1

Install PEEDI firmware version 1.1 (y/[n])? y

WARNING: The firmware image you are trying to load
exceeds your update license. Continue update (y/[n])? y

fis delete peedi
... Erase from 0x01840000-0x019f0000: ..........................
... Erase from 0x019f0000-0x01a00000: .
... Program from 0x007f0000-0x00800000 at 0x019f0000: .
fis create -b 0x100000 -l 0x1AB1C0 -f 0x1840000 -e 0x600040 -r 0x600000
peedi
... Erase from 0x01840000-0x019f0000: ..........................
... Program from 0x00100000-0x002ab1c0 at 0x01840000:
..........................
... Erase from 0x019f0000-0x01a00000: .
... Program from 0x007f0000-0x00800000 at 0x019f0000: .
RedBoot>
```

3.2.3 RedBoot commands used with PEEDI

These commands are used to update, configure, test and run PEEDI:

fconfig

Syntax:

fconfig

Description:

Enter RedBoot and PEEDI configuration parameters

Arguments:

None

Example:

fconfig

update

Syntax:

update [FILEPATH]

Description:

Update PEEDI firmware. If no argument is provided last used will be taken. Default first used argument is taken when **fconfig** command is used.

Arguments:

FILEPATH - filepath of the file

Example:

Update
update http://www.myserver.com/mydir/myfile.bin
update tftp://192.168.1.1/mydir/myfile.bin
update xmodem
update ymodem

memtest

Syntax:

memtest [-c]

Description:

Test available (not occupied by RedBoot) RAM

Arguments:

-c - perform continuous test

Example:

memtest
memtest -c

3.3 Configure PEEDI

3.3.1 Network configuration

RedBoot and PEEDI share the same network settings. To set up the network look in “RedBoot Configuration”.

Note:

Purchased PEEDI is set to get its network settings from a DHCP server.
You can see the PEEDI IP by pressing and holding the green button on the front PEEDI panel. The IP will be shown on the front panel LED indicator. Or connect to PEEDI on the RS232 and the IP is shown during boot-up.

3.3.2 Target configuration file

To operate, PEEDI needs to load a target configuration file, which describes the specifics of the given target; this includes CPU type, FLASH type and metrics, RAM address and size, etc. The target configuration file includes also some settings of PEEDI itself like license keys, baud rates of the serial port, etc.
The target configuration file can be loaded from TFTP, FTP or HTTP server, MMC/SD card or the internal EEPROM. All INI files have standard format: sections are closed in square brackets; comments begin with ‘;’ character and occupy the rest of the line. The configuration file consists of several mandatory sections and others, which can be named freely.
Multiple PEEDIs may load single shared target configuration file, but you need to fill in it all PEEDIs’ licenses.

Section LICENSE

Listed in this section are all the license keys that are acquired, they will unlock specific features of PEEDI.
If there is an FILE parameter in this section, PEEDI will try to load the licenses from this file. The file is searched for an [LICENSE] section containing licenses.

Example:

```
[LICENSE]
KEY = ARM7_ARM9, 1111-2222-3333-4
KEY = UPDATE_29AUG2006, 5555-6666-7777-8
```

```
[LICENSE]
FILE = tftp://192.168.0.1/licenses.txt
```

Section DEBUGGER_PROTOCOL

This section describes the protocol used with the host debugger. Two debugger protocols are supported so far: the GDB Remote and the Angel debug protocol.

REMOTE_PORT

Description

TCP port to be used for accepting connections

Argument

Accepted value 1024 – 65535

PROTOCOL

Description

Describes the debugger protocol. If several protocols need to be enabled, they must be enumerated on the same line, separated by comma.

Argument

gdb_remote – GDB remote protocol will be used
adp – Angel debug protocol will be used

GDB_READ_INGNORE_TIME

Description

This is GDB problem workaround. If read target memory error occurs within this period after the GDB is connected, no error will be returned to GDB. This is because GDB refuses to connect to the target if it gets an error during the connection procedure. GDB tries to read the stack frame during the connection sequence and this may lead to invalid memory accesses.

Argument

Period in milliseconds

Example:

```
[DEBUGGER_PROTOCOL]
PROTOCOL = gdb_remote, adp   ; gdb remote and adp protocols
REMOTE_PORT = 2000           ; TCP/IP port
GDB_READ_INGNORE_TIME = 3000 ; Read error ignore period
```

Section TARGET

This section describes the target's platform.

PLATFORM

Description

Target's platform

Argument

ARM, ARM11, CortexM3, CortexM3_SWD, CortexA8, XScale, Blackfin, MPC5500, MPC8300, ColdFire or MIPS32

Example:

```
[TARGET]
PLATFORM = ARM
```

Section PLATFORM_ARM

In this section there are parameters specific to the given ARM cores. Every core must be defined with the COREn parameter, where 'n' is a number; each parameter related to this core must be preceded with the same COREn prefix. The parameters that must be set are:

Global parameters for all ARM cores:

JTAG_CHAIN

Description

Length of IRs (Instruction Registers) of the devices on the JTAG chain. All IRs must be enumerated; the ones not supported by PEEDI must be skipped when defining COREn parameters (see below)

Argument

Enumerated all JTAG IR lengths, separated by comma

JTAG_CLOCK

Description

JTAG clock before and after initialization

Argument

Both clock values in kHz separated by comma. MAX JTAG clock is 33MHz, but 16-20MHz is recommended. You ca use ADAPTIVE, if the CPU supports adaptive clocking

JTAG_TDO_DELAY

Description

Delay the sample of the TDO JTAG line. For best performance different CPUs require different TDO sample delay. When this parameter is not preset a 5ms value is set by default.

Argument

0 – 35 – the delay in ns.
AUTO – PEEDI tests the CPU and sets the optimum TDO delay

TRST_TYPE

Description

Type of TRST output

Argument

OPENDRAIN or PUSHPULL

RESET_TIME

Description

If 0 is specified, no reset will be issued, this way PEEDI can be attached to already initialized and running target, so INIT section could also be missing. If the target executes code after reset even CORE_STARTUP_MODE=RESET, this means the TAP is not active during reset, add a second argument time argument, this will tell PEEDI to make a second reset pulse after which no code will be executed.

Argument

Period in milliseconds or 0

WAKEUP_TIME

Description

Time to delay the JTAG operations after RESET is released. This time is usually used to allow the clock oscillator to stabilize. You must set it for the Sharp LH7A404, Cirrus EP7209 and the Freescale MAC7100 CPU families.

Argument

Period in milliseconds

WDKICK_TIME

Description

If this parameter is present, PEEDI will periodically kick the TMS470 Analog Watch Dog timer with the specified time between two kicks.

Argument

Period in milliseconds

DBGREQ_OUTPUT

Description

Define the state of the JTAG DBGREQ line

Argument

HIGH, LOW

Core specific parameters:

Note:

Multiple core support is an option that has to be purchased from Ronetix, if the multicore support is not purchased, instead of COREn prefix, where 'n' is integer number, only CORE can also be used.

COREn

Description

Core declaration

Argument

Type of CORE (ARM7TDMI, ARM9TDMI, ARM920T, ARM940T, ARM926E, ARM946E, FEROCEON) and a TAP number separated by comma

COREn_STARTUP_MODE

Description

PEEDI behavior when starting the target:

Argument

RESET	Force the target to debug mode immediately out of reset. No code is executed after reset. (default mode)
STOP, XX	After power-up PEEDI waits XX ms (this gives time to the target to execute its own initialization code) and target is placed in debug mode (halted).
RUN	After reset, the target executes code until stopped by the Telnet "halt" command.

COREn_INIT

Description

Section to be executed in order to initialize the target

Argument

Name of the section

COREn_FLASHm

Description

This parameter points a section which contains the target FLASH description. If multiple FLASH chips/configurations are present on the target each chip/configurations must be described in different section, where 'm' should start from 0 and increment with each new section. If single FLASH chip/configuration is used the 'm' integer number may be skipped. When working with the programmer the first FLASH is selected as current by default. To work on another FLASH, use the **flash set** command to select it. The multiple FLASH support, could also be used to describe different profiles for the same FLASH, for example with different program method type or different image file specified. This way you can easy switch to the desired profile using the **flash set** command

Argument

Name of a section containing FLASH parameters

COREn_ENDIAN

Description

Core endian – little/big

Argument

LITTLE or BIG

COREn_VECTOR_CATCH_MASK

Description

Specifies which interrupts/exceptions to be trapped i.e. break if an interrupt or exception occurs. ARM9 and XSCALE cores only.

Catch vector mask bit meaning:

7	6	5	4	3	2	1	0
FIQ	IRQ	Reserved	D_Abort	P_Abort	SWI	Undefined	Reset

Argument

Value specifying the desired bit mask

COREn_BREAKMODE

Description

Default breakpoint mode. Use to force the usage of hardware break points, when debugging in FLASH, or when working with GDB v5.3, where the **hbreak** command does not work.

Argument

SOFT – software breakpoints will be used

HARD – hardware breakpoints will be used

Note:

The ARM EmbeddedICE logic has hardware resources for two break conditions, never mind break or watch points. The use of software breakpoints allows unlimited number of them, but this still requires the hardware resource of one break/watch point. Software breakpoints are possible only if the code is executed from RAM since the deisred instruction to be breaked on is exchanged with special pattern. In breaf you can use up to two watchpoints or hardware breakpoints; or one watchpoint or hardware breakpoint, and unlimited number of software breakpoints. This means that you may use only one watch point and still debug normally in RAM. But if your code is in ROM/FLASH you must use hardware breakpoints, so if you have set one break or watch point you can still do "signle step", "step in" and "step out", but if you have set two watch or break points, only "continue" is possible after the target breaks, since the debuger needs a temorary break point to achieve the "step" functionality.

COREn_BREAK_PATTERN

Description

Software breakpoint pattern

Argument

32-bit value

COREn_WORKSPACE

Description

Base and length of a region in RAM, used for agent, which allows much faster programming.

Argument

Valid base address within RAM and length in bytes separated by comma

COREn_DATASPACE

Description

Base and length of a region in RAM, used for agent, which allows much faster programming. If this parameter is present, PEEDI will use the workspace for storing only the agent code and the dataspace for the agent data. This is useful when using internal RAM for agent programming, where the internal RAM is code or data only, for example Blackfin CPUs

Argument

Valid base address within RAM and length in bytes separated by comma

COREn_DCC_PORT

Description

TCP port, the target's DCC channel to be routed to.

Argument

1024 to 65535 and optional type of the transfer – 8/32-bit

COREn_PATH

Description

This parameter defines the default path to be used if only a file name (without the full path) is provided to a PEEDI command.

Argument

DEFAULT_PATH

COREn_FILE

Description

This parameter defines the default **memory (multi) load** command's arguments.
This parameter may have two or three arguments. The first argument is the file to be programmed.
The second argument is the file type – BIN, SREC, IHEX or ELF.
The third argument is mandatory for binary files and optional for all other types of files – it is the address where the file should be loaded.

Argument

FILE[, FORMAT][, ADDRESS]

COREn_LOCKOUT_RECOVERY

Description

If this parameter is present, PEEDI automatically executes a "JTAG Lockout Recovery" procedure during reset processing if the MAC7100, STR9 or AVR32 flash is secured.

Argument

YES/NO for STR9 and AVR32 devices
For MAC7100 devices 7-bit value for the CFMCLKD register used during the "JTAG Lockout Recovery". Calculate this parameter based on the reset system clock (PLL disabled). For example:

19 – CLKD for 8MHz system clock
9 – CLKD for 4MHz system clock

COREn_OS

Description

This parameter points to a section which contains parameters that defines the target Operating System. This guides PEEDI to scan the target OS tasks and pass the list to the host debugger.

Argument

A name of a section containing the OS parameters.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_ARM11

This section describes the ARM11 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT and COREn_LOCKOUT_RECOVERY). About the CORE parameter:

COREn

Description

Core declaration

Argument

Type of CORE (ARM1136, ARM1156 or ARM1176) and a TAP number separated by comma. The cores differ by the cp15 registers.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_CortexM3 & Section PLATFORM_CortexM3_SWD

These sections describe the CortexM3 cores connected to PEEDI via JTAG or SWD (Serial Wire Debug). It has all the parameters described in the PLATFORM_ARM section (except the COREn_VECTOR_CATCH_MASK, COREn_BREAK_PATTERN, COREn_DCC_PORT and COREn_LOCKOUT_RECOVERY). The PLATFORM_CortexM3_SWD section has no JTAG_CHAIN parameter, and its clock parameter is named SWD_CLOCK and has the same format as the JTAG_CLOCK parameter. About the CORE parameter:

COREn

Description

Core declaration

Argument

Type of CORE (CortexM3) and a TAP number separated by comma.

COREn_SWO

Description

This parameter is allowed only for the PLATFORM_CortexM3_SWD section.
It tells PEEDI to open a TCP port and listen for incoming telnet connections. PEEDI checks for new incoming telnet connection only when the target CPU is halted.
If a telnet session is opened to that TCP port PEEDI will forward all stimulus data for the given stimulus channel. In order for the CPU to transmit stimulus messages, you need to enable this functionality. This can be done by the target application or by PEEDI using the target INIT script – see chapter ST STM32 family.

Argument

TCP port number, stimulus channel (0 - 31) and optional type of the transfer (8, 16 or 32)

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_CortexA8

This section describes the Cortex-A8 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_VECTOR_CATCH_MASK, COREn_BREAK_PATTERN, COREn_DCC_PORT and COREn_LOCKOUT_RECOVERY). About the CORE parameter:

COREn

Description

Core declaration

Argument

Type of CORE (OMAP3530) and a TAP number separated by comma.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_XSCALE

This section describes the XScale cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, and COREn_LOCKOUT_RECOVERY) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (XScale) and a TAP number separated by comma

COREn _USE_FAST_DOWNLOAD

Description

If YES is specified, PEEDI will send data to the target without checking if the target is ready with the previous data, assuming that the target writes the received data faster than PEEDI is sending it. This type of transfer is faster bun less reliable. Use it only if you are sure that the target is fast enough i.e. the CPU is running on high frequency.

Argument

YES/NO

COREn_DEBUG_HANDLER_ADDR

Description

The address where the XScale debug handler should be mapped at.

Argument

Choosing address has three limitations:

1. Due to the limitation of the ARM branch instruction the address must be within these ranges: 0x00000000 – 0x01FFFC00 or 0xFE000000 – 0xFFFFFC00.
2. Must be aligned to a 1KB (0x400) boundary.
3. Must not overlap user application code.

COREn_VECTOR_UNDEF, COREn_VECTOR_SWI, COREn_VECTOR_PABORT,

COREn_VECTOR_DABORT, COREn_VECTOR_RES, COREn_VECTOR_IRQ,

COREn_VECTOR_FIQ, COREn_RELOCATED_UNDEF, COREn_RELOCATED_SWI

COREn_RELOCATED_PABORT, COREn_RELOCATED_DABORT, COREn_RELOCATED_RES

COREn_RELOCATED_IRQ, COREn_RELOCATED_FIQ

Description

Because of the XScale debugging specifics, PEEDI must be aware of the exception vectors. Each of these parameters must have value of **AUTO** or an exact value which represents a hex encoded ARM instruction. In case of **AUTO** is specified, PEEDI will read the original vector value from the target memory on each debug event (halt, step, go, etc.). Or you can put a constant value if you exactly know the vector's instruction, for example 0xE59ff018 stands for "ldr pc, [pc, #18]" instruction.

Argument

AUTO or a hex encoded ARM instruction.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_MPC5500

This section describes the MPC55XX cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT,

COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (MPC5500) and a TAP number separated by comma

COREn _NEXUS3_ACCESS

Description

This parameter accepts NO or memory region (start address and length in bytes). If a memory region is supplied (usually this is the RAM of the target), PEEDI will access target memory region using the nexus3 module. This method is about three times faster but it uses physical addresses i.e. bypasses the MMU. You can properly use this method if the MMU is set to be transparent i.e. virtual addresses are equal to physical ones.

Argument

NO
START_ADDRESS, LENGTH

COREn _SMALL_REGS_FRAME

Description

From GDB version 6.8, the MPC5500 registers frame has changed. Now latest GDB versions use smaller register frame. For PEEDI to be compatible to the new format set this parameter to YES. If you want it to be compatible to the old GB versions set it to NO.

Argument

YES/NO

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_MPC8300

This section describes the MPC83XX cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT, COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (MPC8313, MPC8321, MPC8323, MPC8343, MPC8349) and a TAP number separated by comma

COREn_BOOT_ADDR

Description

Normally the boot address for PowerPC is 0xFFF00100 or 0x00000100 depending on the Reset Configuration Word (RCW). PEEDI sets a hardware breakpoint at this address to halt the core immediately out of reset.

Argument

The reset vector address.

COREn_RCW

Description

When this parameter is present, PEEDI overrides the Reset Configuration Words with the values provided.

Argument

Reset Configuration Word High, Reset Configuration Word Low

COREn_MMU_PTBASE

Description

This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate the effective address to a physical one using BAT translation, it tries a page translation. For more information see chapter CPU specific considerations.

Argument

Address of the of the pointer to the two page pointers array

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_MPC5200

This section describes the MPC5200 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT, COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (MPC5200) and a TAP number separated by comma

COREn_BOOT_ADDR

Description

Normally the boot address for PowerPC is 0xFFF00100 or 0x00000100 depending on the Reset Configuration Word (RCW). PEEDI sets a hardware breakpoint at this address to halt the core immediately out of reset.

Argument

The reset vector address.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_MPC8500

This section describes the MPC8500 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT, COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (MPC8500) and a TAP number separated by comma

COREn_MMU_TRANS

Description

This parameter sets the default MMU translation address. For example the default Linux kernel address is 0xC0000000.

Argument

Address of virtual address of the memory region which is physically mapped to address 0.

COREn_MMU_PTBASE

Description

This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate

the effective address to a physical one using BAT translation, it tries a page translation. For more information see chapter CPU specific considerations.

Argument

Address of the of the pointer to the two page pointers array

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_PPC400

This section describes the PPC400 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT, COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (PPC405) and a TAP number separated by comma

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_COLDFIRE

This section describes the ColdFire cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the JTAG_CHAIN, JTAG_CLOCK, TRST_TYPE, CORE_ENDIAN, CORE_BREAK_PATTERN, CORE_DCC_PORT, CORE_LOCKOUT_RECOVERY and CORE_VECTOR_CATCH_MASK) including some additional parameters:

BDM_CLOCK

Description

BDM clock before and after initialization

Argument

Both clock values in kHz separated by comma. MAX BDM clock is 33MHz. See your ColdFire CPU user's manual for correct BDM clock. Use ADAPTIVE_n to set the BDM clock to PSTCLK / n

CORE

Description

Core declaration

Argument

Type of CORE – MCF5200, MCF5206, MCF5272, MCF5300, MCF5400

CORE_MEMMAP

Description

Defines a valid memory region. Up to 32 regions can be defined in the target configuration file.

When even one region is defined, PEEDI begins to check every memory access operation if it falls into a defined memory region. If the memory operation is out of the defined regions, PEEDI interrupts the operation and issues an error.

This is made so, because when an access is made to an invalid memory address via the BDM, the ColdFire CPU refuses to respond to any further memory operations until reset.

Argument

Start and end addresses of the desired memory region, separated by comma

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_BLACKFIN

This section describes the Blackfin cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the RESET_TIME, COREn_BREAK_PATTERN,

COREn_DCC_PORT, COREn_LOCKOUT_RECOVERY and COREn_VECTOR_CATCH_MASK) including some additional parameters:

COREn

Description

Core declaration

Argument

Type of CORE (BF522, BF525, BF527, BF531, BF532, BF533, BF534, BF535, BF536, BF537, BF538, BF539, BF542, BF544, BF548, BF549, BF561A, BF561B) and a TAP number separated by comma

The following parameters are not mandatory. They are used to define a "virtual" memory region corresponding to an external memory mapped device that is bigger than the visible external asynchronous memory space. The higher address lines of the device that are not connected to the CPU address buss must be driven by the GPIO pins. This way you can use all the PEEDI CLI commands (**flash program**, **memory read**, etc.) on the defined virtual region as the whole device is directly visible in the memory space of the target. Actually PEEDI emulates this behavior by accessing physically the device only through the memory window provided by the CPU external address space and driving the higher address lines of the device depending on the address that is requested to be accessed.
This feature of PEEDI helps programming FLASH chips which are bigger than the visible external asynchronous memory space.

COREn_VMEM

Description

Defines a memory region, which is virtually mapped to big external memory mapped device.

Argument

Start address and length in bytes of the virtual memory region.

COREn_VMEM_WINDOW

Description

Defines a memory window to physically access the external memory mapped device, t.e. the memory region where the device is mapped into the CPU memory space.

Argument

Start address and length in bytes of the memory window.

COREn_VMEM_PINS

Description

List of the GPIO pins connected to the higher external device address lines that are not connected to the CPU address bus, starting from lowest to highest, separated by comma. The GPIO pins must belong to the same GPIO port.

Argument

Pxy, where 'x' is the GPIO port A..J and 'y' is the bit number 0..15.

Imagining that we want to virtually map on address 0x30000000, an 8MB FLASH that is connected to the first chip select of the CPU, so physically accessible at 0x20000000 via 1MB window and its A19, A20 and A21 pins are connected to PF4, PF5 and PF8 CPU pins, the configuration should look like this:

```
; The 8MB FLASH is virtually mapped at 0x30000000
CORE0_VMEM              = 0x30000000, 0x800000

; It is physically visible through a 1MB window at 0x20000000
CORE0_VMEM_WINDOW       = 0x20000000, 0x100000

; PF4, PF5 and PF8 are used to drive A19, A20 and A21 of the FLASH
CORE0_VMEM_ADDRESS_PINS= PF4, PF5, PF8
```

Now we can erase, program and verify the whole 8MB of FLASH at address 0x30000000 using any PEEDI **flash** command. Keep in mind that when defining [FLASH] section in the target configuration file, you need to specify the virtual address of the FLASH for the BASE_ADDR parameter.

CORE_MEMMAP

Description

Defines a valid memory region. Up to 32 regions can be defined in the target configuration file.
When even one region is defined, PEEDI begins to check every memory access operation if it falls into a defined memory region. If the memory operation is out of the defined regions, PEEDI interrupts the operation and issues an error.
This is made so, because when an access is made to an invalid memory address via the JTAG, the Blackfin CPU may stop to respond to any further memory operations until reset.

Argument

Start and end addresses of the desired memory region, separated by comma

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_MIPS32

This section describes the MIPS32 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT and COREn_LOCKOUT_RECOVERY). About the CORE parameter:

COREn

Description

Core declaration

Argument

Type of CORE (MIPS32 or PIC32) and a TAP number separated by comma.

For example of this section see the example configuration files in the appendix of this document.

Section PLATFORM_AVR32

This section describes the AVR32 cores connected to PEEDI. It has all the parameters described in the PLATFORM_ARM section (except the COREn_BREAK_PATTERN, COREn_DCC_PORT and COREn_VECTOR_CATCH_MASK). About the CORE parameter:

COREn

Description

Core declaration

Argument

Type of CORE (AVR32AP7 or AVR32UC3) and a TAP number separated by comma.

COREn _BLOCK_ACCESS

Description

This parameter accepts NO or memory region (start address and length in bytes). If a memory region is supplied (usually this is the RAM of the target), PEEDI will access target memory region using the MEMORY_WORD_ACCESS TAP command.

Argument

NO
START_ADDRESS, LENGTH

For example of this section see the example configuration files in the appendix of this document.

Section INIT

This is the section specified by COREn_INIT parameter. It includes commands, which are executed once after every target power detection and target reset. The purpose of this section is to initialize the target (map the memory, init peripherals and so on). Most of these are **memory write** commands.

Example:

```
[INIT_EB55800]
memory write 0xFFFF4020 0x004F0002 ; enable main clock
wait 100                           ; wait to stabilize
memory write 0xFFFF4020 0x004F4002 ; switch to main clock
memory write 0xFFFF4020 0x3F006802 ; enable PLL
wait 100                           ; wait to lock
memory write 0xFFFF4020 0x3F008722 ; switch to PLL, pres=4, mul=8

memory write 0xFFE00020 0x00000001 ; cancel reset remapping
memory write 0xFFE00000 0x010020A5 ; csr0 - Flash at 0x1000000, 2 Ws
memory write 0xFFE00004 0x02003029 ; csr1 - RAM  at 0x2000000, 2 Ws
```

Sometimes it is impossible to initialize the target only by using the commands in the [INIT] section of the target configuration file. In cases like this to perform the initialization an executable image can be loaded and executed in the target using the **memory load** and **go** commands. Before loading the image, the RAM where it will be loaded must be initialized. Follow these steps to make a successful initialization:

Note:
This is working [INIT] section for AT91M55800A CPU. In this case the last instruction of the executable must be SWI, informing that job has finished.

```
[INIT_EB55800]
; First init chip selects
memory write 0xFFE00020 0x00000001 ; cancel reset remapping
memory write 0xFFE00000 0x010020A5 ; csr0 - Flash at 0x1000000, 2 Ws
memory write 0xFFE00004 0x02003029 ; csr1 - RAM  at 0x2000000, 2 Ws
memory write 0xFFFFF124 0xFFFFFFFF  ; disable all interrupts

; Then load and start the executable
memory load tftp://192.168.1.1/init.bin bin 0x20
             ; load executable image, skipping interrupt table
set cpsr 0xD3 ; set supervisor mode, interrupts disabled
set sp 0x200  ; set stack pointer, if program uses stack
breakpoint add 0x8  ; set break at software interrupt vector
                    ; address
go            ; start executable
wait 50       ; wait to complete
halt          ; halt if not completed
break del 1   ; del break at software interrupt vector address
```

Section FLASH

This section tells PEEDI what type are the onboard FLASH chips and what their configuration is. These are all possible variants of connecting different FLASH chips:

- Internal microcontroller FLASH
or external:
- One 8-bit chip, forming 8-bit architecture
- Two 8-bit chips, forming 16-bit architecture
- Four 8-bit chips, forming 32-bit architecture
- One 16-bit chip, forming 16-bit architecture
- Two 16-bit chips, forming 32-bit architecture
- One 32-bit chip, forming 32-bit architecture

When describing external FLASH configuration the following parameters must be specified:

External FLASH:
CHIP
CHECK_ID

ACCESS_METHOD
CHIP_WIDTH
CHIP_COUNT
BASE_ADDR
FILE
AUTO_ERASE

Considering your configuration you must specify CHPIS, and WIDTH parameters, WIDTH is the width of a single chip, so system width will be CHIPS multiplied by WIDTH.

When describing internal microcontroller FLASH, the FLASH section parameters depends on the specific microcontroller family.

For Atmel AT91SAM7 family these are:

Atmel AT91SAM7:
CHIP
CPU_CLOCK
AUTO_LOCK
SECURE_FLASH
FILE
AUTO_ERASE

For Atmel AVR32UC3 family these are:

Atmel AVR32UC:
CHIP
AUTO_LOCK
SECURE_FLASH
FILE
AUTO_ERASE

For Philips LPC2000 family these are:

Philips LPC2000:
CHIP
CPU_CLOCK
CHECK_ID
FILE
AUTO_ERASE
SET_VECTORS_CHECKSUM

For Philips LPC2900 family these are:

Philips LPC2900:

CHIP
CPU_CLOCK
FILE
AUTO_ERASE
ACCESS_METHOD

For Freescale MAC7100 family these are:

Freescale MAC7100:
CHIP – MAC7100 or MAC7100_DATA
CPU_CLOCK
FILE
AUTO_ERASE

For ST STR7 family these are:

ST STR7:
CHIP
DATA_BANK
ACCESS_METHOD
BASE_ADDR
SECURE_FLASH
FILE
AUTO_ERASE
AUTO_LOCK

For ST STM32 family these are:

ST STM32:
CHIP
BASE_ADDR
ACCESS_METHOD
FILE
AUTO_ERASE

For Microchip PIC32 family these are:

PIC32:
CHIP
ACCESS_METHOD
FILE
AUTO_ERASE

For TI TMS470 family these are:

TI TMS470:
CHIP
ACCESS_METHOD
BASE_ADDR
PROTECTION_KEY0
PROTECTION_KEY1
PROTECTION_KEY2
PROTECTION_KEY3
ALLOW_ZERO_KEYS
FILE
AUTO_ERASE

For Luminary LM3S family these are:

LM3S:
CHIP
CPU_CLOCK
ACCESS_METHOD
FILE
AUTO_ERASE

For SPI FLASH or Atmel DataFlash family, connected to an Atmel AT91 CPU these are:

SPI or AT45:
CHIP
CPU
SPI_DIV
nSPI
nCS
SPI_SPCK
SPI_MISO
SPI_MOSI
SPI_CS
FILE
AUTO_ERASE

For SPI FLASH or Atmel DataFlash family, connected to a BlackFin CPU these are:

SPI or AT45:
CHIP
CPU
SPI_DIV
SPI_CS

FILE
AUTO_ERASE

For SPI FLASH FLASH or Atmel DataFlash family, connected to a NXP LPC2000 CPU these are:

SPI or AT45:
CHIP
CPU
SPI_DIV
CS_ASSERT
CS_RELEASE
FILE
AUTO_ERASE

For NAND FLASH devices these are:

NAND:
CHIP
CPU
DATA_BASE
CMD_BASE
ADDR_BASE
CS_ASSERT
CS_RELEASE
ALE_ASSERT
ALE_RELEASE
CLE_ASSERT
CLE_RELEASE
BAD_BLOCK_TABLE
BAD_BLOCKS
ERASE_BAD_BLOCKS
SWAP_BI
OOB_INFO
FILE
AUTO_ERASE
DAVINCI_UBL_DESCIPTOR_MAGIC
DAVINCI_UBL_DESCIPTOR_ENTRY_POINT
DAVINCI_UBL_DESCIPTOR_LOAD_ADDR
DAVINCI_UBL_MAX_IMAGE_SIZE

For OneNAND FLASH devices these are:

OneNAND:

CHIP
ADDR_BASE
BAD_BLOCKS
ERASE_BAD_BLOCKS
OOB_INFO
FILE
AUTO_ERASE

For MPC5500 devices these are:

MPC5500:
CHIP
BASE_ADDR
ALLOW_SHADOW_MODIFY
FILE
AUTO_ERASE

For MCF5200 devices these are:

MCF5200:
CHIP
BASE_ADDR
IPS_BASE
CPU_CLOCK
FILE
AUTO_ERASE

CHIP

Description

FLASH chip type. To find if your flash is supported and see its exact name, use **flash find**. If your FLASH chip is not supported by the current FLASH database please contact us and we will provide you with the latest database. This parameter may be present multiple times in a single FLASH section, each time specifying different FLASH chip. This way if the CHECK_ID parameter is YES, PEEDI will read the onboard FLASH ID and will find the right chip among the all chips enumerated using the CHIP parameter.

Argument

Flash chip present in FLASH DB

CHECK_ID

Description

When specified YES, if single FLASH chip is described by the CHIP parameter, PEEDI will check if the onboard FLASH chip reports the same as selected by the CHIP parameter. If multiple FLASH chips are enumerated using more than one CHIP parameter, PEEDI will automatically consider the chip which ID matches the reported by the onboard FLASH chip.

Argument

YES/NO

ACCESS_METHOD

Description

Flash programming method. If AGENT is specified, the FLASH programmer will return an error if the agent failed to start; if AUTO is specified, the programmer will try to start the agent; if failed it will perform direct programming. If DIRECT is specified the programmer will perform direct programming.

Note:

Programming using agent is many times faster than programming directly. To enable agent usage set COREn_WORKSPACE parameter in the PLATFORM section of the target configuration file.

Argument

AUTO/AGENT/DIRECT

CHIP_WIDTH

Description

Chip width, some FLASH chips support several widths.

Argument

8/16/32

CHIP_COUNT

Description

Number of FLASH chips.

Argument

1/2/4

BASE_ADDR

Description

Start address of FLASH.

Argument

First valid FLASH address.

FILE

Description

This parameter defines the default **flash (multi) program** command's arguments.
This parameter may have two or three arguments. The first argument is the file to be programmed.
The second argument is the file type – BIN, SREC, IHEX or ELF.
The third argument is mandatory for binary files and optional for all other types of files – it is the address where the file should be loaded.

Argument

FILE_NAME, FILE_FORMAT[, FILE_ADDRESS]

AUTO_ERASE

Description

Do or do not erase affected FLASH sectors before program operation, for more information, see **flash program** command.

Argument

YES or NO

AUTO_LOCK

Description

Do or do not lock affected FLASH sectors (if supported from FLASH) after program operation, this would prevent FLASH form accidental write or erase operations.

Argument

YES or NO

CPU_CLOCK

Description

The CPU clock after the initialization.
Used when describing internal FLASH of Atmel AT91SAM7, Philips LPC2000 and FreeScale MAC7100 or MCF5200 series microcontrollers

Argument

Frequency in kHz

SECURE_FLASH

Description

Do or do not secure FLASH to avoid external reading operations. This disables the JTAG interface until the whole FLASH is erased, so any JTAG operations are impossible after FLASH is programmed and secured.
Used when describing internal FLASH of Atmel AT91SAM7 series microcontrollers.

Argument

YES or NO

SET_VECTORS_CHECKSUM

Description

Set this parameter to YES, if you want PEEDI to automatically calculate and set the exception vectors checksum at address 0x14 while programming FLASH. This check sum is required by the microcontroller bootloader as evidence that valid user application resist in the FLASH, so the control will be passed to it.
Used when describing internal FLASH of Philips LPC2000 series microcontrollers.

Argument

YES or NO

DATA_BANK

Description

Set this parameter to YES, if your device has flash data bank (bank1)
Used when describing internal FLASH of ST STR7 series microcontrollers.

Argument

YES or NO

PROTECTION_KEY0 – PROTECTION_KEY3

Description

These four parameters define the four FLASH security keys that are used to unlock the FLASH for erasing and writing.
Used when describing internal FLASH of TI TMS470 series microcontrollers.

Argument

C-style 32bit HEX value

ALLOW_ZERO_KEYS

Description

This parameter is used to prohibit programming of new Memory Security Module keys that are all 0x00000000, because this will permanently lock the TMS against debugging and programming.
Used when describing internal FLASH of TI TMS470 series microcontrollers.

Argument

YES or NO

CPU

Description

Target CPU
Used when describing NAND, SPI or Atmel DataFlash.

Argument

AT91RM9200, AT91SAM9261, AT91SAM9263, AT91SAM7, iMX21, iMX25, iMX27, iMX31, iMX35, iMX51, BF5XX, BF52X, BF54X, MPC83XX, TMS320DM355, LPC2XXX, LPC318X_MLC, PXA3XX

SPI_DIV

Description

SPI divider:
AT91RM9200: Fspi = (MCK/2)/SPI_DIV
AT91SAM9261: Fspi = MCK/SPI_DIV
Used when describing Atmel DataFlash.

Argument

Divider value

nSPI

Description

SPI controller to use
Used when describing Atmel DataFlash.

Argument

0 or 1

nCS

Description

Chip select to use
Used when describing Atmel DataFlash.

Argument

0 to 3

SPI_SPCK

Description

This describes which PIO is dedicated to the SPI SPCK signal
Used when describing Atmel DataFlash.

Argument

controller, peripheral, pin, for example:
PIOA, A, 2 - PIOA controller, peripheral A, pin 2

SPI_MISO

Description

This describes which PIO is dedicated to the SPI MISO signal
Used when describing Atmel DataFlash.

Argument

controller, peripheral, pin, for example:
PIOA, A, 0 - PIOA controller, peripheral A, pin 0

SPI_MOSI

Description

This describes which PIO is dedicated to the SPI MOSI signal
Used when describing Atmel DataFlash.

Argument

controller, peripheral, pin, for example:
PIOA, A, 1 - PIOA controller, peripheral A, pin 1

SPI_CS

Description

This describes which PIO is dedicated to the SPI CS signal
Used when describing Atmel DataFlash.

Argument

controller, peripheral, pin, for example:
PIOA, A, 3 - PIOA controller, peripheral A, pin 3
If the CPU parameter is set to BF5XX, this parameter accepts values of 1 to 7, which corresponds to FLG1-FLG7. See BlackFin's SPI_FLG.

CMD_BASE

Description

Base address, that if written to, the NAND CLE signal will be asserted.
On MPC83XX devices with built-in NAND FLASH controller this parameter tells PEEDI the offset of Internal Memory Mapped Registers, i.e. value of IMMRBAR.

Argument

Address

DATA_BASE

Description

Base address, that if written to, the NAND ALE and CLE signals will be inactive.

On MPC83XX devices with built-in NAND FLASH controller this parameter tells PEEDI the address of the data buffer used by NAND FLASH controller.

Argument

Address

ADDR_BASE

Description

Base address, that if written to, the NAND ALE signal will be asserted.

Argument

Address

CS_ASSERT

Description

Describes memory write operation ([address]=data) that will assert the NAND chip select connected to a PIO pin.

Argument

Address and data to be written

CS_RELEASE

Description

Describes memory write operation ([address]=data) that will release the NAND chip select connected to a PIO pin.

Argument

Address and data to be written

ALE_ASSERT

Description

Describes memory write operation ([address]=data) that will assert the NAND Address Latch Enable connected to a PIO pin.

Argument

Address and data to be written

ALE_RELEASE

Description

Describes memory write operation ([address]=data) that will release the NAND Address Latch Enable connected to a PIO pin.

Argument

Address and data to be written

CLE_ASSERT

Description

Describes memory write operation ([address]=data) that will assert the NAND Command Latch Enable connected to a PIO pin.

Argument

Address and data to be written

CLE_RELEASE

Description

Describes memory write operation ([address]=data) that will release the NAND Command Latch Enable connected to a PIO pin.

Argument

Address and data to be written

BAD_BLOCK_TABLE

Description

If this parameter is set to YES, PEEDI will check for Linux style main and mirror Bad Block Tables and if not found, it will create them on the last two good blocks of the NAND FLASH chip.

Argument

YES/NO

BAD_BLOCKS

Description
List of blocks to be marked as bad.
Argument
Block numbers separated by comma

ERASE_BAD_BLOCKS
Description
If this parameter is se to YES, PEEDI will try to erase even the bad NAND blocks.
Argument
YES/NO

WARNING:
If you erase blocks factory marked as bad, there is no way to detect which were the bad blocks.

SWAP_BI
Description
If this parameter is se to YES, PEEDI will swap the bad block marker ECC byte with a spare one. This option is applicable for iMX21, iMX25, iMX27, iMX31 and iMX35 targets only.
Argument
YES/NO

OOB_INFO
Description
How to deal with the Out Of Band (OOB/spare) page bytes. Spare bytes are extra bytes added to the page.

Argument

JFFS2	– data bytes will be read from the image file, spare bytes will be filled with ECC data (6 bytes for 512 bytes page, 24 bytes for 2048 bytes page).
JFFS2_NO_EM	– like JFFS2 but PEEDI does not write erase/clean markers
RAW	– data and spare bytes will be loaded from the image file, default if OOB_INFO parameter is missing
YAFFS	– like RAW, but bad blocks are skipped
FF	– only data bytes will be read from the image file, spare bytes will be set to 0xFF
AT91SAM9	– Atmel AT91SAM9 hardware ECC
BLACKFIN_ECC	– ADI Blackfin hardware ECC
IMX_ECC	– Freescale iMX hardware ECC
LPC_ECC	– NXP LPC hardware ECC
OMAP3_ECC	– TI Omap3 hardware ECC
PXA_ECC	– Marvell PXA3XX hardware ECC
ONENAND	– OneNAND hardware ECC
DAVINCI_ECC	– TI DaVinci Hardware ECC, 4 bytes per 512 bytes data
DAVINCI_ECC_HW6_512	– DaVinci hardware ECC, 6 bytes per 512 bytes data
DAVINCI_ECC_HW6_512_2610	– DaVinci hardware ECC, 6 bytes per 512 bytes data with workaround for a JFFS2 bug in Linux kernel 2.6.10
DAVINCI_ECC_HW10_512	– DaVinci hardware ECC, 10 bytes per 512 bytes data

DAVINCI_UBL_DESCIPTOR_MAGIC
Description
Descriptor magic – the first 32-bit value in the UBL descriptor.
It set to non-zero value, programming of the file image is relocated with one NAND Flash page (512 or 2048 bytes). The skipped page is used for the UBL descriptor and it is filled by PEEDI.
Used when describing NAND FLASH for TI DaVInci CPU.
Argument
Value

DAVINCI_UBL_DESCIPTOR_ENTRY_POINT

Description

This value will be programmed at offset 0x4 in the UBL descriptor
Used when describing NAND FLASH for TI DaVInci CPU.

Argument

Value

DAVINCI_UBL_DESCIPTOR_LOAD_ADDR

Description

Used when describing NAND FLASH for TI DaVInci CPU.

Argument

Value

DAVINCI_UBL_MAX_IMAGE_SIZE

Description

Used by PEEDI to print a warning if the programmed file size exceeds this limit.
Used when describing NAND FLASH for TI DaVInci CPU.

Argument

Value

ALLOW_SHADOW_MODIFY

Description

This parameter is used to prohibit programming/erasing of Shadow FLASH memory, because this may permanently lock the MPC5500 device against debugging and programming.
Used when describing internal FLASH of Freescale MPC5500 microcontrollers.

Argument

YES or NO

IPS_BASE

Description

Internal Peripheral System base address

Argument

IPS address

SERIAL_NUM

Description

If this parameter is present, PEEDI will program a unique serial number on the given FLASH location with each **flash program** command, this way if PEEDI is used in production, each board programmed will get an unique serial number.
The parameter has tree arguments:
FILE – path to a text file which contains the last serial number that is programmed. The file must contain only one line with the number, with no leading trailing spaces or any other characters. For example:
<start of file>
341
<end of file>
Each time PEEDI programs a board, it loads the file, gets the last serial number increments it and stores the new value back in the file, and so if the file resides on a TFTP or a FTP server, the server must allow write access (upload) of the file. File may also resides on a MMC/SD card or in the PEEDI internal EPPROM file system, note that each EEPROM has a limited number of writes.
ADDRESS – where the serial number will be programmed, must be aligned to the serial number bits width
WIDTH – bits width of the serial number value – 16, 32, 64
Currently the SERIAL_NUM parameter is supported for external NOR FLASH chips, Atmel DataFlash SPI chips and Atmel AT91SAM7 devices.

Arguments

FILE, ADDRESS, WIDTH

Section OS

This section contains parameters which help PEEDI to scan the target OS task list.

BASE

Description

Size and address of the first task item in the item list

Argument

Size of the item pointer in bytes and the address of it separated by comma, the address may be represented by a path to a file (ex. *tftp://192.168.1.1/task_base*), this way PEEDI will dynamically get the base from external file that could be auto generated by the makefile using command like this **nm vmlinux | grep –w init_task > /tftproot/task_base**. This way you don't need to change the PEEDI CFG every time the base address is changed from build to build.

NEXT

Description

Size and offset of the field in the task structure that contains the address of the next task structure in the task list

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

NEXT_THREAD

Description

Size and offset of the field in the task structure that contains the address of the next thread of the multithreaded task in the task list

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

PID

Description

Size and offset of the field in the task structure that contains the unique task ID

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

NAME

Description

Size and offset of the field in the task structure that contains the task name

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

CODE_START

Description

Size and offset of the field in the task structure that contains the address of the task .code section

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

DATA_START

Description

Size and offset of the field in the task structure that contains the address of the task .data section

Argument

Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

BSS_START

Description
Size and offset of the field in the task structure that contains the address of the task .bss section
Argument
Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

CONTEXT
Description
Size and offset of the field in the task structure that contains saved task context
Argument
Size of the filed in bytes and the offset of it from the beginning of the task structure separated by comma

CONTEXT_REG
Description
Using this parameter multiple times, you can define which registers are saved in the task context
Argument
Name (as listed by **info registers** PEEDI command) and offset of a register saved in the task context separated by comma

Section SERIAL

The serial interface gets its configuration parameters from this section. These parameters are:

BAUD
Description
Baud rate
Argument
1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200

STOP_BITS
Description
Stop bits
Argument
1, 1.5 or 2

PARITY
Description
Parity
Argument
NONE, EVEN, ODD

TCP_PORT
Description
Port, serial traffic to be routed to. If set to 0, the PEEDI serial port is used for command line interface
Argument
1024 – 65535, 0 – use PEEDI serial for command line interface
Example:

```
[SERIAL]
BAUD=115200
STOP_BITS=1
PARITY=NONE
TCP_PORT=2023
```

Section TELNET

This section has only two parameters. The first sets the new command prompt string after the configuration file is loaded. The second parameter can be omitted; it tells what ASCII code to be used for backspace action.

PROMPT

Description

This will change the default PEEDI telnet prompt

Argument

The prompt, closed in quotation marks

BACKSPACE

Description

Telnet backspace character ASCII code

Argument

The code

Example:

```
[TELNET]
PROMPT = "peedi> "
BACKSPACE = 127
```

Section DISPLAY

These sections parameters specify the brightness of the seven segment LED indicator and the volume of the speaker, both accept values in the range 0 - 100

BRIGHTNESS

Description

Seven segment LED indicator brightness

Argument

0 – 100

VOLUME

Description

Speaker volume

Argument

0 – 100

Example:

```
[DISPLAY]
BRIGHTNESS=20
VOLUME=51
```

Section ACTIONS

Declares what scripts can be executed using front panel buttons, each declaration must be on a new line. The declaration consists of a number associated with the specified script name. A section with the same name must exist somewhere in the target configuration file. If AUTORUN=N parameter is specified, where N is number of a script, the given script will be executed every time a target is connected to PEEDI. For more information see Script execution using the front panel interface .

Example:

```
[ACTIONS]
AUTORUN = 2
1 = erase_program_verify
2 = prog_http

[erase_program_verify]
```

```
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000
flash verify tftp://192.168.1.41/main_romram.bin bin 0x400000

[prog_http]
flash prog http://192.168.1.41/main_romram.bin bin 0x400000 erase
```

3.4 CPU specific considerations

WARNING:
The following may be extremely important for your target so read it very carefully.

3.4.1 Atmel AT91SAM7 family

The Atmel AT91SAM7 family devices need a special FLASH section in the target configuration file, for more info rmation see Section FLASH and Appendix A: Sample target configuration files.

3.4.2 Atmel AVR32UC family

The Atmel AVR32UC family devices need a special FLASH section in the target configuration file, for more info rmation see Section FLASH and Appendix A: Sample target configuration files.

3.4.3 Philips LPC2000 family

The Philips LPC2000 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

To successfully connect to a LPC2000 device the pull-down resistor that enables the JTAG interface must not be more than 1k, because PEEDI has internal 10k pull-ups.

Because the JTAG clock is synchronized to the internal CPU clock it is recommended to use adaptive JTAG clock or clock up to 1MHz for normal work (the second argument of the JTAG_CLOCK parameter).

To successfully program a LPC2000 device make sure you have specified valid RAM address for the CORE_WORKSPACE parameter in the PLATFORM_ARM section. The internal RAM starts from 0x40000000, so this is a good value for this parameter.

To successfully verify the FLASH contents, first you must set the MEMMAP register to map the flash vectors at address 0x00000000 like this:

memory write 0xE01FC040 0x00000001

You may issue the previous command every time you need to verify or you may put it in the init section of the core in the target configuration file, this way it will be executed automatically.

To secure the LPC2000 device, your application must set FLASH address location 0x1FC (User flash sector 0) with value 0x87654321 (2271560481 Decimal) when programmed. This will disable the JTAG port and some of the ISP commands on the next reset.

The only way to un-secure the device is to use ISP command to erase the FLASH. This can be made with the Philips LPC2000 FLASH utility.
For more information about the LPC2000 securing (code protection) read the LPC2000 user manual.

3.4.4 Freescale MAC7100 family

The Freescale MAC7100 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

In program FLASH the address range from 0x0400 to 0x041F is used to set the FLASH security. If there is user code this could secure the FLASH incidentally, so avoid placing code there.

flash erase chip – perform MASS ERASE
flash lock – write at address 0xFC100414 = 0xFFFFFFFC0 (enable flash security)

If the Flash security is enabled, the only way to unlock the device is to perform JTAG Lockout Recovery procedure.

PEEDI executes a “JTAG Lockout Recovery” during reset processing if the MAC7100 flash is secured and if the configuration file contains:
CORE_LOCKOUT_RECOVERY = clkd

3.4.5 ST STR7 family

The ST STR7 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

In the STR7 microcontrollers, several memory regions may be mapped at address 0x00000000, including the internal FLASH. So these are the acceptable values for the BASE_ADDR parameter in the FLASH section:
- 0x00000000 (STR71x and STR73x)
- 0x40000000 (STR71x)
- 0x80000000 (STR73x),

If the SECURE_FLASH target configuration parameter is set to YES. The first time, the device is secured by programming the DBGP bit of the NVAPR0 register. Each time after, the device is secured programming the next un-programmed bit PEN bit of the NVAPR1 register.

Keep in mind that once secured, the device may be temporary or permanently unsecured only by the code that is programmed in the FLASH, so avoid securing the device if the code inside it can not unlock it, because the devise may become unusable.

The device can be permanently secured-unsecured only sixteen times, because after that all NVAPR1 bits are programmed.

3.4.6 ST STR9 family

The ST STR9 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

You can program the FLASH configuration and security using this PEEDI raw JTAG commands manually or put in a script:

```
; ISC_ENABLE
jtag pui 17 lsb 0x1FF0C idle          ; IR = ISC_ENABLE
jtag rd 10 lsb 0x3FF ilde             ; DR = 2 bypass+8 dont care bits

; prog config
jtag ri 17 lsb 0x1FF11 idle           ; IR = ISC_ADDRESS_SHIFT
jtag rd 10 lsb 0x50 ilde              ; DR = Configuration register
jtag ri 17 lsb 0x1FF20 idle           ; IR = ISC_PROGRAM
jtag rd 32 lsb 0x00000000 stop        ; write 32 config bits
jtag 0 32 lsb 0x00000000 stop         ; write 32 config bits
jtag 0 2 lsb 0x3 idle                 ; write 2 bypass bits
wait 1000                             ; wait ready

; prog secure
jtag ri 17 lsb 0x1FF11 idle           ; IR = ISC_ADDRESS_SHIFT
jtag rd 10 lsb 0x80 ilde              ; DR = Configuration register
jtag ri 17 lsb 0x1FF22 idle           ; IR = ISC_PROGRAM_SECURITY
jtag rd 2 lsb 0x3 idle                ; skip the two bypass bits
wait 1000                             ; wait ready

; ISC_DISABLE
jtag ri 17 lsb 0x1FF0C idle           ; IR = ISC_DISABLE
jtag rd 10 lsb 0x3FF ilde             ; DR = 2 bypass+8 don’t care bits
```

If the Flash security is enabled, the only way to unlock the device is to perform JTAG Lockout Recovery procedure. PEEDI executes a "JTAG Lockout Recovery" during reset processing if the STR9 flash is secured and if the configuration file contains:
CORE_LOCKOUT_RECOVERY = YES

3.4.7 ST STM32 family

The ST STM32 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

In the STM32 microcontrollers, the FLASH may be write-protected. The protection is set using the STM32 option bytes. Option bytes are used to configure also other STM32 CPU settings – for more information see the STM32F10xxx Flash programming.

For managing STM32 option bytes, PEEDI has the **flash this option** command. To erase all option bytes use '**flash this option erase**'.

To write a single option byte, use '**flash this option BYTE VALUE**'. An option byte can be written only once after it is erased. If you want to change the value of previously written byte you must erase it – this will erase all other option values, so you may need to set them again.

You can see current option bytes using '**memory read16 0x1FFFF800 8**'.

Use the following commands in the target INIT script to enable SWO stimulus output:

```
; init SWO
mem write 0xE0042004 0x20          ; enable async trace
mem write 0xE0040010 1             ; SWO prescaler 1
mem write 0xE00400F0 1             ; enable Manchester encoding
mem write 0xE0040304 0             ; bypass formatter
mem write 0xE0000FB0 0xC5ACCE55    ; unlock access to ITM registers
mem write 0xE0000E80 0x10009       ; trace ID = 1, ITM enabled
mem write 0xE0000E40 0xF           ; enable all tracing ports
mem write 0xE0000E00 0xFFFFFFFF    ; enable all stimulus ports
```

PEEDI supports only Manchester SWO encoding up to 66MHz.
PEEDI checks for new incoming telnet connection only when the target CPU is halted.
If the SWO functionality seems unstable, lower the CPU clock or increase the SWO prescaler, both of these will result in lower SWO clock.

3.4.8 TI TMS470 family

The TI TMS470 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

TMS470 devices use four WORD long keys to protect FLASH from unwanted erase/write operations, so be careful not to write them accidentally. The keys are reported every time the FLASH is programmed.

Every time PEEDI first tries to unlock FLASH using the default keys (0xFFFFFFFF), if fails it uses the keys pointed out in the target configuration file. This way you can erase and program the FLASH without the need of changing the keys for each operation, because during FLASH erase the keys are automatically set to 0xFFFFFFFF.

In TMS470 devices that have Memory Security Module (MSM), if the currently programmed MSM keys are different from 0xFFFFFFFF, you have to put this unlock sequence in the INIT section:

```
[INIT_TMS470]
; dummy read the four keys
m r 0x0000ffe0
m r 0x0000ffe4
m r 0x0000ffe8
m r 0x0000ffec
```

```
; try to unlock the device using the correct MSM keys
m w 0xFFFFF700 0xXXXXXXXX
m w 0xFFFFF704 0xXXXXXXXX
m w 0xFFFFF708 0xXXXXXXXX
m w 0xFFFFF70C 0xXXXXXXXX
```

Where 0xXXXXXXXX's are the right MSM keys. The four word passwords location in the internal FLASH for the MSM1 is placed starting from the last eight words of the first flash sector. Please see the datasheet of your TMS470 CPU to check the right addresses of the keys in FLASH memory and the addresses of the registers where the keys have to be entered. These passwords are used to insecure the device in case it has been partly secured.

The ALLOW_ZERO_KEYS FLASH section parameter is used to protect the device from unwanted permanent locking of the device – this may happen if MSM keys all of 0x000000000 are programmed in to the FLASH.

Some TMS470 devices have internal Analog Watch Dog timer (AWD). The AWD must be disabled in order to use PEEDI for debugging or programming. The AWD can be disabled by grounding the AWD pin. Alternatively WDKICK_TIME CFG parameter can be used and PEEDI will kick periodically the AWD.

3.4.9 Intel XScale family

Debugging XScale core is a little complicated by the fact that the exception vectors must be cached in the mini instruction cache where the debug handler resides. PEEDI provides two ways of defining the vectors (see Section PLATFORM_XSCALE). First is to set them fixed – suitable when the vectors are not updated dynamically at runtime. And the second is to tell PEEDI to read them from the target's memory each time a debug event occurs – suitable when vectors are set by the user application at runtime. There are several ways to provoke a debug event:

1. Set 32 bit write access watch point at the last modified by the user code vector.
2. Set hardware breakpoint to a point of the code where the vectors have been set but not yet enabled.
3. In the source code, add a software break "*asm("bkpt 1");*", where the vectors have been set but not yet enabled. PEEDI recognizes this special break and immediately starts the target again with refreshed vectors.

Once the target has been stopped by the desired debug event, you can again start it and the exception vectors will be updated. You can use the first way and a **wait** command to automate this in the INIT section of the target configuration file:

```
[INIT_XSCALE]
break add watch 0xffff001C w 32    ; set watchpoint on FIQ vector
go                                  ; start target
wait 30000 stop                     ; wait to break
go                                  ; start again with updated vectors
```

If the vectors are set during code download, they will be automatically updated if defined AUTO.
If you want the target to be stopped after it is initialized, just remove the last two lines from the previous example of the XScale init section.

Avoid using code that potentially could invalidate the mini instruction cache during debug:

- Don't use "**MCR p15, 0, rd, c7, c5, 1**", instead use "**MCR p15, 0, rd, c7, c5**"
- Disable **CONFIG_XSCALE_CACHE_ERRATA** (Workaround for XScale cache errata) option when building Linux kernel.

3.4.10 NAND FLASH

PEEDI is able to program all NAND chips with 8 and 16 bits bus and up to 8Gbits of size.

Generally the NAND Flash programming is CPU independent, but the given target must comply with the following hardware requirements:
The CPU must have a direct access to the NAND Flash chip:

1. To be able to write/read data from/to the NAND Flash with one memory transaction command.
2. To be able to write ADDRESSES and COMMANDS (ALE and CLE) to the NAND Flash with one memory transaction command.
3. To be able to ASSERT and RELEASE the chip select, ALE and CLE of the NAND Flash with one memory write command

PEEDI doesn't use the Read/Busy signal.

The INIT section of the config file must include the initialization for the chip select and the GPIOs, because the Flash Programmer doesn't make any initialization.

The NAND Flash devices may have blocks that are invalid when they are shipped.
An invalid block is one that contains one or more bad bits. Additional bad blocks may develop with use.
The factory identifies invalid blocks before shipping by programming data other than FFh (x8) or FFFFh (x16) into the first spare location of the first or second page of each bad block. PEEDI automatically detects the bad blocks and reports them using the **flash info** and **flash query** commands.

Once detected, the bad blocks are protected against erasing and programming.
On demand, PEEDI can be forced to try to erase the existing bad blocks.
It is also possible to force blocks as bad.

To erase all blocks including the bad blocks, set the ERASE_BAD_BLOCKS parameter to YES. After PEEDI restart, the command "flash erase" will erase all blocks.

WARNING:
If you erase blocks factory marked as bad, there is now way to detect which were the bad blocks.

Make sure you have saved the output of the “flash query” command so you can mark again the bad blocks as bad.

To force marking of blocks 4, 27 and 1002 as bad set BAD_BLOCKS parameter like this:

```
BAD_BLOCKS = 4, 27, 1002
```

After PEEDI restart, the "flash info" command will mark the given blocks as bad. Once marked as bad, the blocks a not marked anymore.

PEEDI supports direct programming of JFFS2 images to the NAND flash. For this, the OOB_INFO parameter must be set to ‘JFFS2’. This way PEEDI will the write the data loading from the image file and will calculate the ECC and program it to the OBB/spare bytes. PEEDI supports only BIN images starting from address 0. When programming the image bad blocks will be just skipped and left un-programmed. They will not affect the block count order.

See at91sam9261_pm9261.cfg and pxa270_custom.cfg from our example configuration:

Appendix A: Sample target configuration files

If you use a custom file system, using PEEDI you can program a bootloader to the NAND chip, that will gain the control of the system after it is rebooted and could handle the programming of the left empty NAND FLASH chip, considering the NAND file system you use and the bad block present in the given target.

WARNING:
The PEEDI TFTP client uses 512 bytes or 2048 bytes (if supported by the TFTP server) transfer block size, which limits the size of the image file to 32MB or 128MB. If your file is bigger, use HTTP/FTP file server or use MMC/SD card to store the file and put it on PEEDI.

3.4.11 Freescale MPC5500 family

The Freescale MPC5500 family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

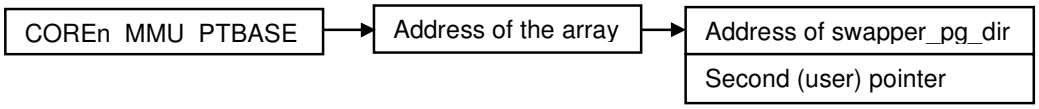
The ALLOW_SHADOW_MODIFY FLASH section parameter is used to protect the device from unwanted permanent locking – this may happen if Shadow FLASH is erased or the configuration word programmed to something different than 0x55AA55AA. For more information read MPC55XX Reference Manual.

Insight/gdb versions below 6.0 are not supported, because they have different register order defined in the gdb remote debug protocol.

3.4.12 Freescale PowerQUICC II Pro MPC83XX family

PEEDI supports debugging Linux kernel running on MPC8300 devices with MMU enabled. If the instruction or data address translation is enabled (IR and DR bits set in MSR), PEEDI assumes all the addresses used by gdb or the user in the console to be effective and need to be translated to physical ones before the very memory access. First PEEDI ties a BAT translation, if it fails and the COREn_MMU_PTBASE parameter is present in the target configuration file, then PEEDI tries a page translation on the given address.

The COREn_MMU_PTBASE parameter must point to a physical address which contains the virtual address of the two pointers array:



Before debugging the user must manually set all the pointers using **mem write** commands which can be put in the INIT section for example.

Newer Linux kernels have built-in support for automatic update of the pointers, so no user setup is required, only the COREn_MMU_PTBASE must be set to 0xF0, where the kernel puts the pointer to the array. But this feature must be enabled in order to be used – "make menuconfig"→*kernel hacking*→ *Include BDI-2000 user context switcher*, here you can also set the *Compile the kernel with debug info* option.

Sometimes gdb and insight do not want to load the debug info because of an internal bug, in this case adding "-gstabs+" option in the makefile fixes this.

WARNING:

On some cores (MPC8349) in order the software breakpoints to work, the interrupt vectors must reside in valid memory. So you must either initialize the memory properly or either set the MSR[IP] to a value so the vectors fall on a valid memory.

When the COREn_RCW CFG parameter is present, PEEDI overrides the Reset Configuration Words with the values provided. This is useful when the RCW that is fetched during board reset does not suit the user's needs when debugging or programming the board using PEEDI.
The COREn_BOOT_ADDR CFG must be set considering the RCW that is fetched/set, because RCW sets the reset vector address (the boot address).
If PEEDI reports "++ info: target does not enter debug mode, forcing halt", this might mean that the CPU boots from an address different than the one set by the COREn_BOOT_ADDR parameter. So you should check again both COREn_RCW and COREn_BOOT_ADDR parameters.

3.4.13 Freescale MCF5200 family

The Freescale MCF52xx family devices need a special FLASH section in the target configuration file, for more information see Section FLASH and Appendix A: Sample target configuration files.

WARNING:

Set very carefully the CPU_CLOCK parameter otherwise the FLASH may be damaged.

3.4.14 Analog Devices Blackfin family

Most of the Blackfin devices can access only up to 4MB of external FLASH memory. Usually this limitation is workarounded by connecting the free higher address pins of FLASH to PIO PFx pins of the CPU. So the software running on the CPU must ensure the PFx pins are driven properly to access the

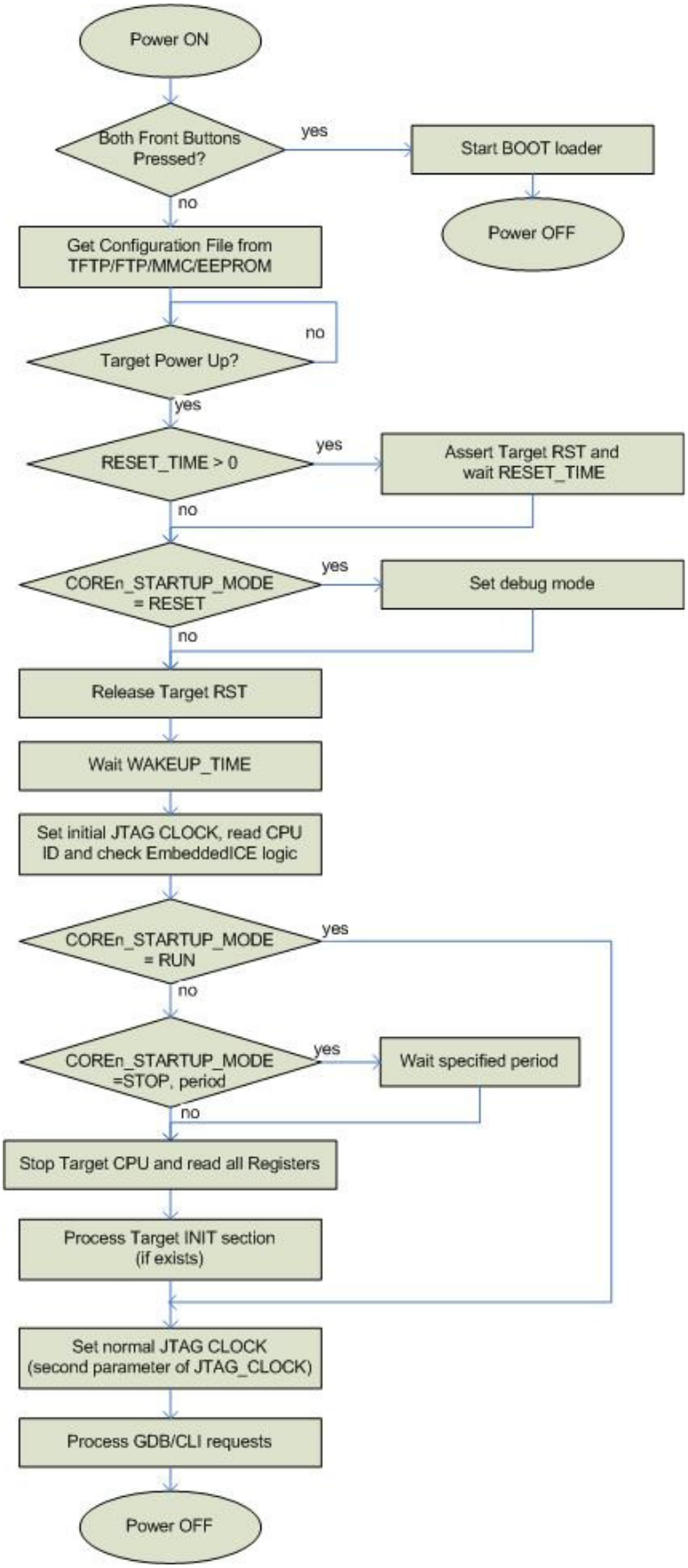
desired part of the FLASH. PEEDI also supports this kind or FLASH configuration making the work with such kind of configuration just like as the FLASH is entirely visible. For more information see Section PLATFORM BLACKFIN.

PEEDI supports programming of NAND flash chips connected to the CPU's async bus or the NFC controller. FLASH and Atmel DataFlash chips connected to the CPU's SPI. For all these configurations PEEDI writes only to the specific peripheral controller registers. It is the user's responsibility to set the needed GPIOs in the INIT section of the CFG file, i.e. to set the corresponding PORTx_FAR and PORTx_MUX registers so the FLASH is accessible through async bus, NFC or SPI. For help on this check the sample CFG files from our website.

It is observed that the presence of the ADI USB JTAG debug interface (BF535+Spartan FPGA), interferes the normal PEEDI operation (seen with some EZ-KIT and STAMP boards). Some BlackFin boards do not work reliably with low JTAG clock, so if you experience problems in the INIT section, please use 2MHz init JTAG clock.

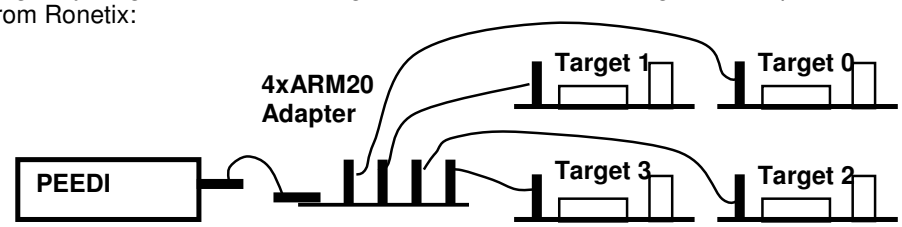
3.5 Boot sequence

On power-up if the front panel buttons are pressed, the bootloader is started. If not, PEEDI tries to load the configuration file. After that it checks if the target is powered. Then if the RESET_TIME is bigger than 0 it asserts the target RST and waits the specified time. After that if COREn_STARTUP_MODE is set to RESET, PEEDI sets the target in debug mode, which assures that no instructions are executed when the target RST is released. Next the target RST is released and PEEDI waits the WAKEUP_TIME. After that the initial JTAG clock is set. If the COREn_STARTUP_MODE is RUN it switches to the normal JTAG clock and the target is left running. Else it checks if the COREn_STARTUP_MODE is STOP and if yes it waits the specified period and stops the target. Then the init section is processed. After that the JTAG clock is switched to its normal speed.
Note that Different cores may have different COREn_STARTUP_MODE parameter set.
The following diagram shows the boot sequence:


Power ON
Both Front Buttons Pressed?
yes
Start BOOT loader
Power OFF
no
Get Configuration File from TFTP/FTP/MMC/EEPROM
Target Power Up?
no
yes
RESET_TIME > 0
yes
Assert Target RST and wait RESET_TIME
no
COREn_STARTUP_MODE = RESET
yes
Set debug mode
no
Release Target RST
Wait WAKEUP_TIME
Set initial JTAG CLOCK, read CPU ID and check EmbeddedICE logic
COREn_STARTUP_MODE = RUN
yes
no
COREn_STARTUP_MODE =STOP, period
yes
Wait specified period
no
Stop Target CPU and read all Registers
Process Target INIT section (if exists)
Set normal JTAG CLOCK (second parameter of JTAG_CLOCK)
Process GDB/CLI requests
Power OFF

3.6 Multiple core support

The multiple core (extra license is required) support allows you to debug/program up to 4 independent targets using only single PEEDI. The targets must be chained using the multiple core cable adapter available from Ronetix:



WARNING:

All targets must have equal power supply (10% tolerance is permissible). The highest power supply is taken for reference for the PEEDI output schematic, so the JTAG signals will have that value.

WARNING:

You must use as short as possible cables, because the equivalent cable length is the sum of all cables. Even then, you may need to decrease the JTAG clock in the target configuration file.

The JTAG_CHAIN parameter in the PLATFORM_ARM section must be correctly set and each core must be described.

Example PLATFORM_ARM section describing two chained cores:

```
[PLATFORM_ARM]
JTAG_CHAIN = 4, 4       ; list of IR length of all TAP controller in
                        ; the JTAG chain
JTAG_CLOCK = 10, 12500 ; JTAG Clock in [kHz] - 10 kHz JTAG clock for
                        ; init operations and 12.5MHz for normal work
TRST_TYPE = OPENDRAIN   ; type of TRST output: OPENDRAIN or PUSHPULL
STARTUP_TIME = AUTO     ; wait until target reset pulse finished
RESET_TIME = 20         ; length of RESET pulse in ms

CORE0 = ARM7TDMI, 0                ; TAP 0 is ARM7TDMI CPU
CORE0_STARTUP_MODE = STOP, 300     ; Stop 300ms after reset
CORE0_INIT = INIT_CORE0            ; init section for this core
CORE0_BREAKMODE = soft             ; use software breakpoints
CORE0_BREAK_PATTERN = 0xDFFFDFFF   ; software breakpoint pattern
CORE0_FLASH0 = FLASH_CORE0         ; flash section for this core
CORE0_ENDIAN = LITTLE              ; core endian
CORE0_WORKSPACE = 0xC00000, 0xE00 ; workspace for FLASH programmer

CORE1 = ARM9TDMI, 1                ; TAP 1 is ARM9TDMI CPU
CORE1_STARTUP_MODE = RESET         ; Stop immediately
CORE1_INIT = INIT_CORE1            ; init section for this core
CORE1_BREAKMODE = soft             ; use software breakpoints
CORE1_BREAK_PATTERN = 0xDFFFDFFF   ; software breakpoint pattern
CORE1_FLASH0 = FLASH_CORE1         ; flash section for this core
CORE1_ENDIAN = LITTLE              ; core endian
CORE1_VECTOR_CATCH_MASK = 0x10     ; catch data abort exceptions
CORE1_WORKSPACE = 0xA00000,0x2600 ; workspace for FLASH programmer
```

Each core has its own TCP port number to wait for a debug session. The port number is the number specified in the DEBUGER section of the target configuration file plus the number of the core. For example if the port specified is 2000 and the core number is 2 (starting from 0), then you should connect to PEEDI for a debug session at TCP port 2002:

```
(gdb) target remote 192.168.1.10:2000 // first target
(gdb) target remote 192.168.1.10:2002 // third target
```

Note:
The reset JTAG signal is common for all targets, so if one developer resets his target, all targets will get reset.

When opening a CLI telnet session, the first core (number 0) is selected as default core. To select another core to work with, use the **core** command:

```
Peedi> core #1
```

When debugging via the Angel debug protocol, using the **adp** CLI command you can choose which core to connect to:

```
Peedi> adp #1
```

Now of you connect to PEEDI using the ADP protocol you will debug core 1.

Using the **flash multi program** and **flash multi erase** CLI commands, you can program up to four targets at once, saving huge amounts of time when many boards need to be programmed:

```
peedi> flash multi erase #0 #1
peedi> flash multi program #0 #1 tftp://192.168.1.1 myfile elf
```

This will program targets 0 and 1 simultaneously.

3.7 Script execution using the front panel interface

You can define various command scripts in the configuration file and execute them using the front panel buttons. Press the green button to choose the script you wish to execute, the LED indicator will show the numbers associated with the available scripts, when ready with the choice press the red button to start the script. While the script is being executed the LED indicator will rotate its segments to show that the execution is in progress. When the script is successfully completed, the led indicator will show the chosen script and the speaker will produce a single beep notifying the end of operation. If an error occurs during execution (some of the commands exited with error code), the execution is terminated, the LED indicator will start to blink with the error code, and the speaker will beep a number of times equal to the error code. Then you can start the script again by pressing the red button. If you press the green button the display will show the current script, pressing it again will show the next available script, so you can chose another script to execute. Here are the available error codes:

1 TIMEOUT

2 NOT FOUND

3 INVALID ARGUMENT

4 GENERIC ERROR

More information about the error can be obtained by connecting to PEEDI using telnet and then restarting the script. Status messages are output to every opened telnet connection when a script is executed.
The scripts are defined in the following manner: in the [ACTIONS] section list all scripts that you want to define in this format:
N = script_name
where N a hex number (1-9 and A-F) and is associated with the script_name, then define section named [script_name] and put any number of commands, each command must be on a new line.
These scripts are useful when using PEEDI in autonomous (stand-alone) mode, not connected to a PC. In such mode PEEDI can be used as a stand-alone FLASH programmer. If all needed files are stored on a MMC/SD card no Ethernet cable is necessary and PEEDI will need only power supply cable. If AUTORUN=N parameter is specified, where N is number of a script, the given script will be executed

every time a target is connected to PEEDI. This eliminates the need to manually start the script, very useful and time saving when great volumes of targets are to be programmed.
Example:

```
[ACTIONS]
AUTORUN=3
1 = prog_tftp_dump
2 = erase_program
3 = prog_tftp
4 = prog_card
6 = dump_tftp
8 = dump_card

[erase_program]
flash erase 0x400000 0x600000
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000

[prog_tftp]
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000 erase

[prog_tftp_dump]
flash prog tftp://192.168.1.41/dump.bin bin 0x400000 erase

[prog_card]
flash prog card://dump.bin bin 0x400000 erase

[dump_tftp]
memory dump 0x400000 0x100000 tftp://192.168.1.41/dump.bin

[dump_card]
memory dump 0x400000 0x100000 card://dump.bin
```

3.8 Serial Interface

PEEDI's RS232 connector is routed to a predefined TCP port (in the configuration file). This way if a telnet connection is opened to that TCP port, the telnet application will receive each byte coming in through the RS232 port. Vice versa, all data that is sent from the telnet application to the TCP port is forwarded to the RS232 port. Note that no flow control is supported. You can use the normal command line telnet connection to PEEDI simultaneously, as they work completely independent.

For information how to set serial port parameters, see section SERIAL in "Target configuration file" chapter.

3.9 ARM DCC Interface

On ARM targets, PEEDI routes the core's DCC to a TCP port. This way if a telnet connection is opened to that TCP port, the telnet application will receive each byte coming in through the DCC.

The TCP port to connect to, is the value specified for the COREn_DCC_PORT parameter in the target configuration file.

You can use these GNU GCC compatible, simple C functions in your target code to communicate via the DCC:

```
#define DCC_TX_BUSY  2
#define DCC_RX_READY 1

unsigned int dcc_recv_char( void )
{
      unsigned int cc, status;
```

```
      do    __asm__ volatile ( "mrc p14,0, %0, c0, c0\n" : "=r" (status));
      while ( !(status & DCC_RX_READY) );

      __asm__( "mrc p14,0, %0, c1, c0\n" : "=r" (cc));

      return cc;
}

void dcc_send_char( unsigned int cc )
{
      unsigned int status;

      do    __asm__ volatile ( "mrc p14,0, %0, c0, c0\n" : "=r" (status));
      while ( status & DCC_TX_BUSY );

      __asm__( "mcr p14,0, %0, c1, c0\n" : : "r" (cc));
}

void dcc_send_string( const char* ss )
{
      while ( *ss ) dcc_send_char( *ss++ );
}
```

Keep in mind that these are blocking functions.

3.10 Working with Insight/gdb

To be able to debug an application with gdb the application must be compiled using the '-g –O0' options to enable debugging and disable optimizing.

When your application is built and ready to be debugged, start gdb or insight:

```
$ arm-elf-gdb myapp
```
or
```
$ arm-elf-insight myapp
```

To connect to the target (assuming that your PEEDI is set to use IP 192.168.1.10) type in the console window:

```
(gdb) target remote 192.168.1.10:2000
```

This will tell GDB to connect to PEEDI using remote protocol. Now you can load your application into target's memory like this:

```
(gdb) load
```

This will load required application sections into target memory at addresses specified during the link process. User can manage these addresses using linker script files. While load command is being executed, gdb sets PC to the entry point of the application, if you want to start execution from another point or just the real entry point is different from the one set by gdb, you can manually set PC to a desired location like this:

```
(gdb) set $pc=0x200040
```

If you want to make sure that your application starts with all interrupts disabled, you can do this:

```
(gdb) set $cpsr=0xD3
```

If your application utilizes stack and the startup code does not initialize the stack pointer you can do this manually like this:

```
(gdb) set $sp=0x201000
```

Now your application is ready to be debugged:

```
(gdb) continue    ; start the application
```

or

```
(gdb) si          ; make single step
```

When finished debugging, you can leave gdb/insight in to ways – with or without resetting the target. To exit resetting the target type:

```
(gdb) quit
```

Otherwise type

```
(gdb) detach
(gdb) quit
```

To make your life easier you may define various commands in a gdb init file and tell gdb to load that file when starting like this:

```
$ arm-elf-insight --command=my_gdb_init
```

Assuming that your PEEDI has IP 192.168.1.10, my_gdb_init file may contain something like this:

```
# this will tell gdb to connect to PEEDI using remote protocol
target remote 192.168.1.10:2000
info target
# the following will define a user command
define ll
     set $cpsr=0xD3
     load
end
```

Download Performance

By default, GDB versions previous than v6.5.50 (from 06.08.2006) download program code and data in small packets that are not necessarily a multiple of four bytes in length. This causes program download times to be slower than necessary, especially with ARM targets. There are two GDB internal variables that affect this. To improve GDB download performance, you should set the download write size to a binary value 16KB and the memory write packet size to a larger value to allow for packet overhead (+100 bytes is plenty). For example, to download 16KB at a time:

```
(gdb) set remote memory-write-packet-size fixed
(gdb) set remote memory-write-packet-size 16384
(gdb) set download-write-size 16128
```

Note:

No error is returned to GDB in case of an invalid 4 byte memory read access during a period defined by ***GDB_READ_INGNORE_TIME*** *in the target configuration file. This is because GDB refuses to connect to the target if it gets an error during this connection sequence. GDB tries to read the stack frame during the connection sequence and this may lead to invalid memory access.*

3.11 Working with other debuggers

Since PEEDI supports Angel Debug Protocol (ADP), any debugger which can connect to the ARM's Angel Debug Monitor may be used to debug with PEEDI.

At the moment most of the debuggers are ADP compatible so you may use practically all of them to debug with PEEDI. Generally you must set the debugger to connect to PEEDI (which will act as a target debug monitor) via Ethernet, using the Angel debug monitor protocol on the IP you have assigned to your PEEDI.

Since ADP is not designed to be used with multiple cores, if your PEEDI is connected to more than one core, you can select the core you wish to connect to, using the **adp** CLI command.

Make sure you have enabled the ADP protocol in the PEEDI's configuration file:

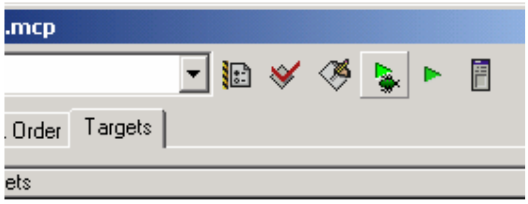
```
[DEBUGGER]
PROTOCOL = adp                  ; ADP debug protocol
```

Here are some examples on how to set some of the most popular debuggers to work with PEEDI:

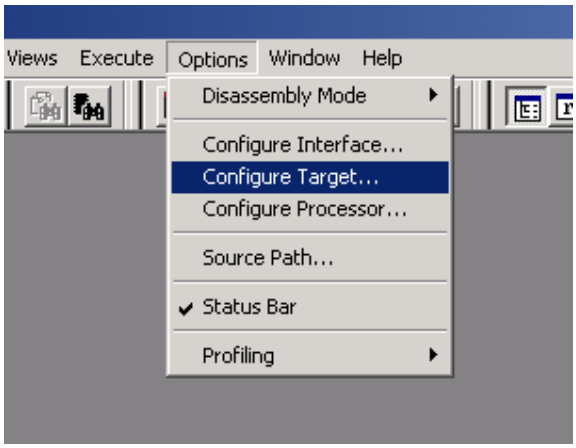
Note:
To successfully connect to PEEDI using the ADP protocol, the ADP protocol must be enabled in the target configuration file of PEEDI and its license to be present as well. For more information see the explanation of the DEBUGGER section of the target configuration file in the PEEDI user's manual.

3.11.1 ARM Developer Suite (ADS) v1.2, AXD debugger

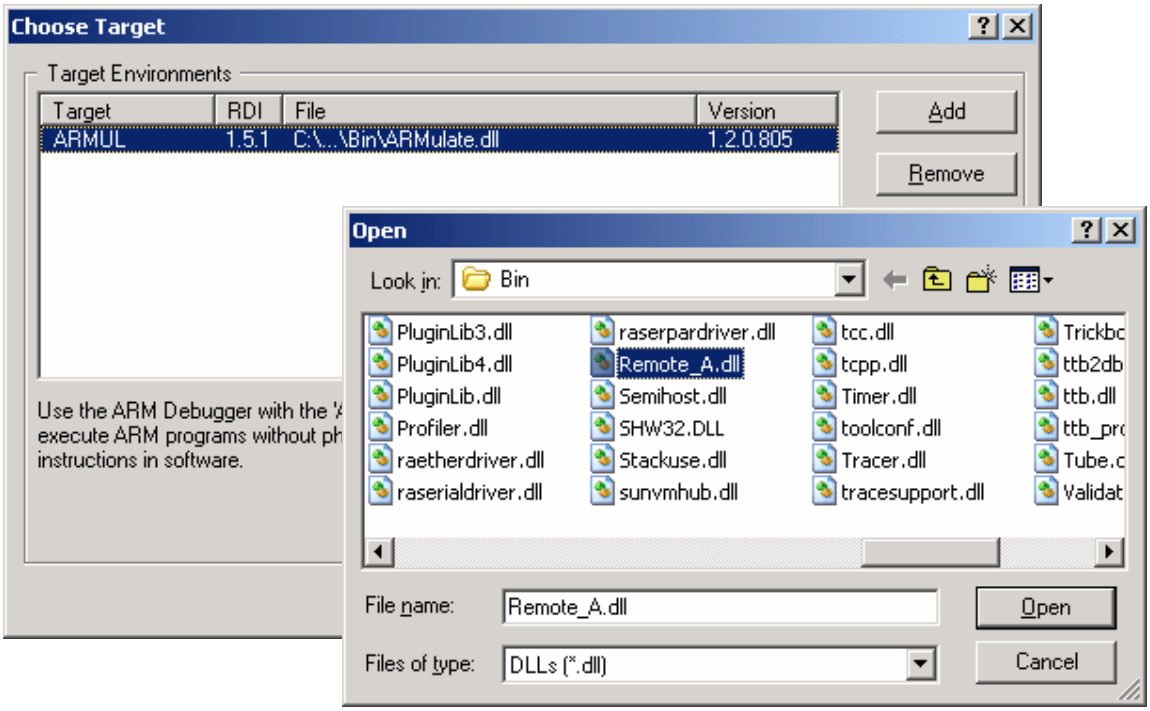
When your project is built, click the debug Icon to start the AXD debugger



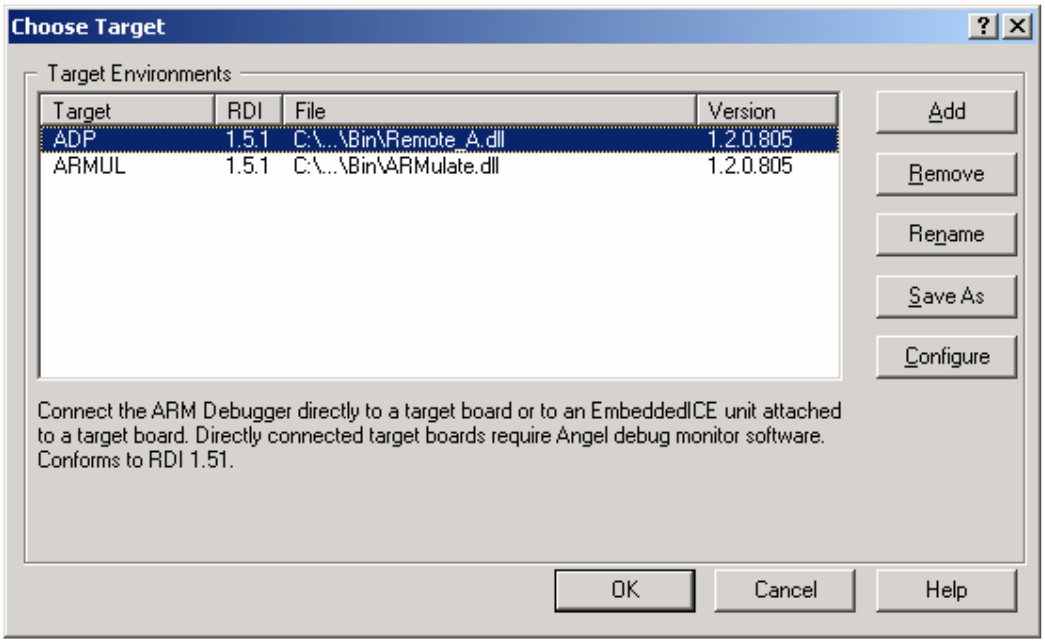
Then click Options->Configure Target… to open the Choose Target dialog:



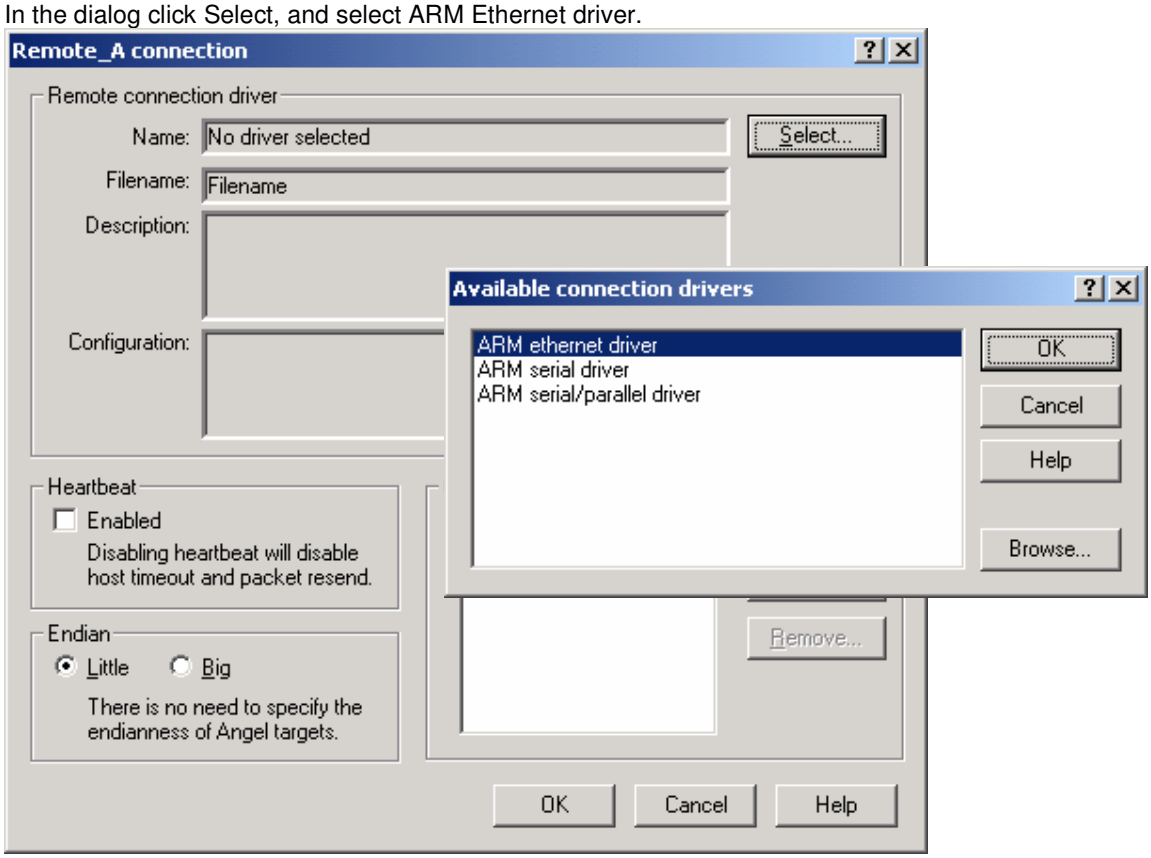
Here click Add to open the Open file dialog. Select the Remote_A.dll in the ADS's Bin folder and click Open. Remote_A.dll is the RDI-to-ADP interface library.



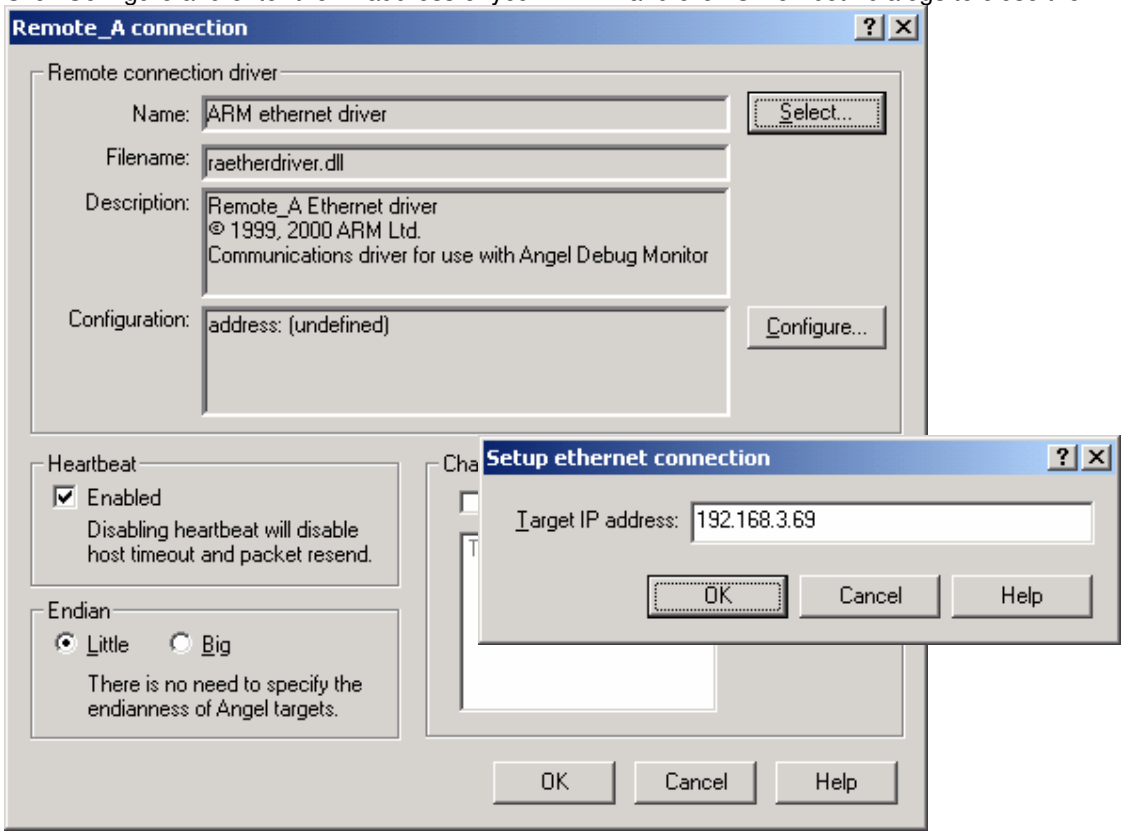
Now select the new created target and click Configure to open the Remote_A connection dialog



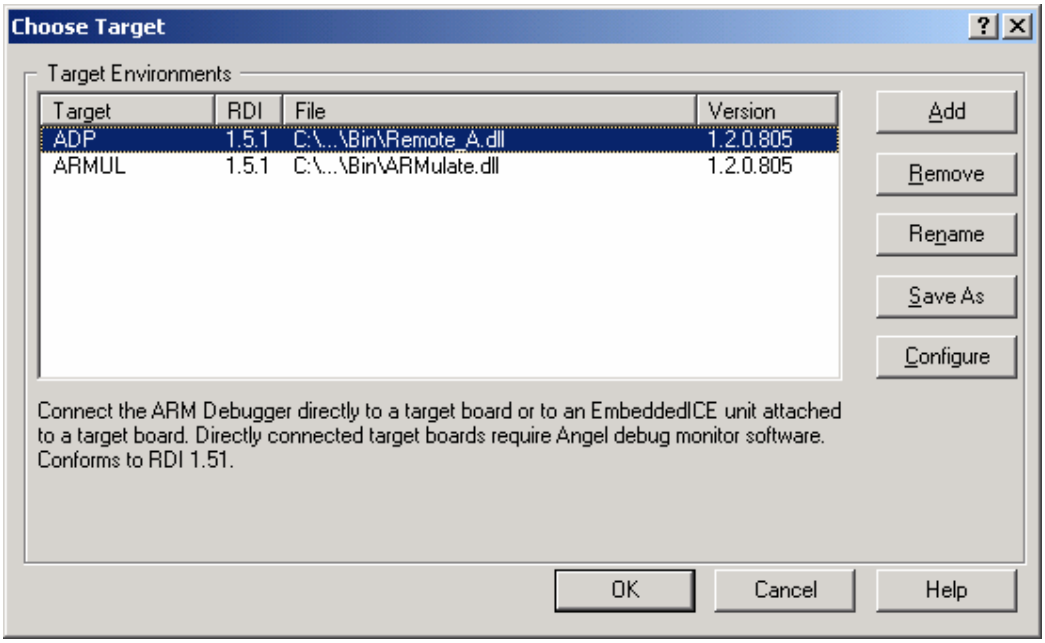
In the dialog click Select, and select ARM Ethernet driver.



Click Configure and enter the IP address of your PEEDI and click OK on both dialogs to close them:



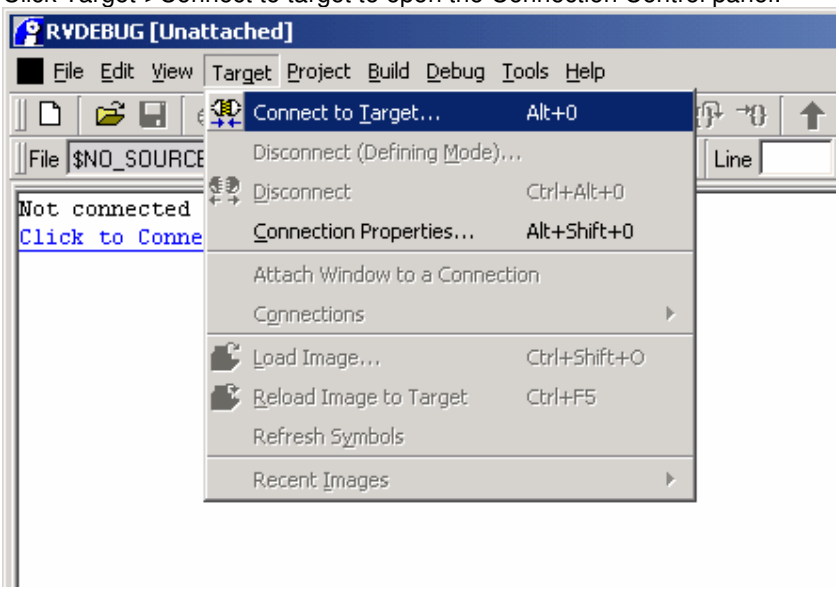
After that click OK on the Choose target dialog and the debugger should connect to PEEDI:



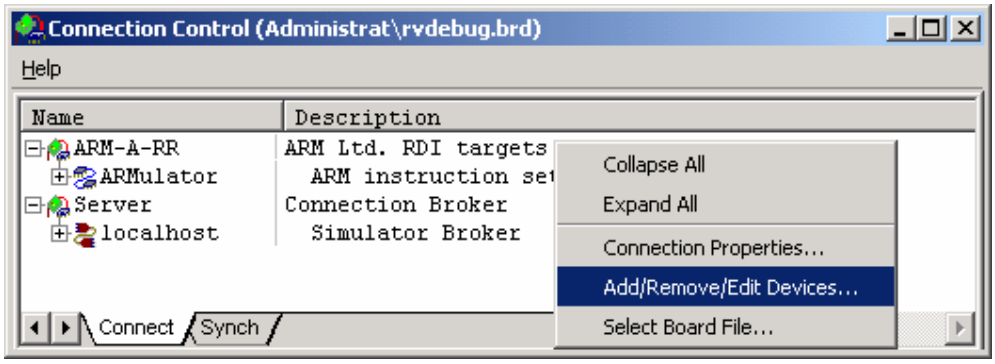
At this point you should be able to debug your project.

3.11.2 ARM RealView Developer Suite (RVDS) v2.2, RealView debugger

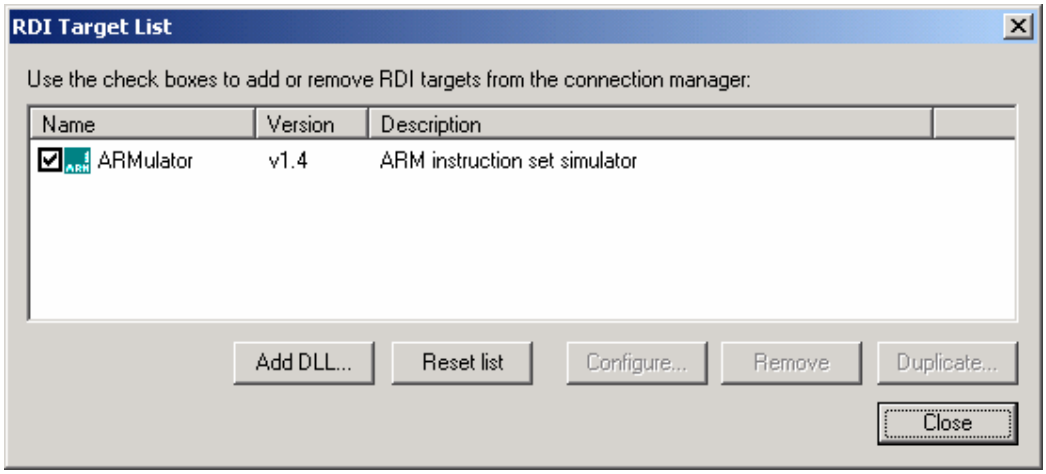
Click Target->Connect to target to open the Connection Control panel:



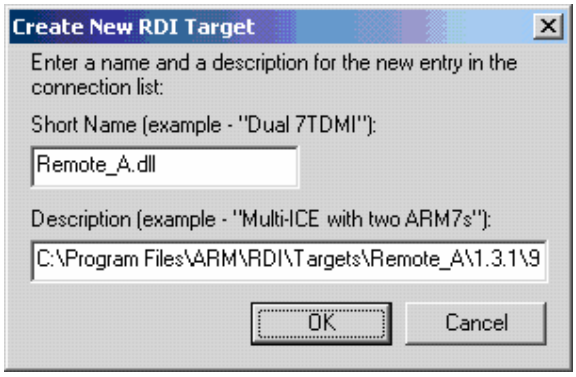
In the panel, right-click on the ARM-A-RR item and next click Add/Remove/Edit devices… to open the RDI target list:



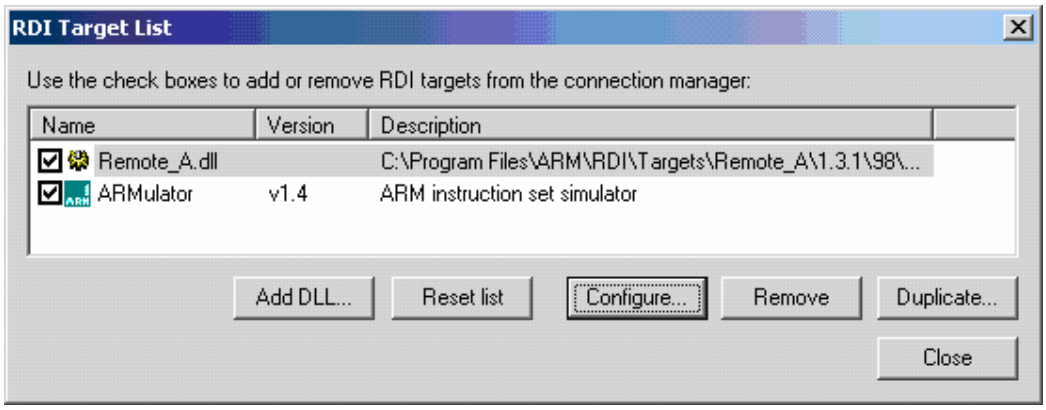
Here click Add DLL, in the appeared dialog locate and select the Remote_A.dll RDI library.



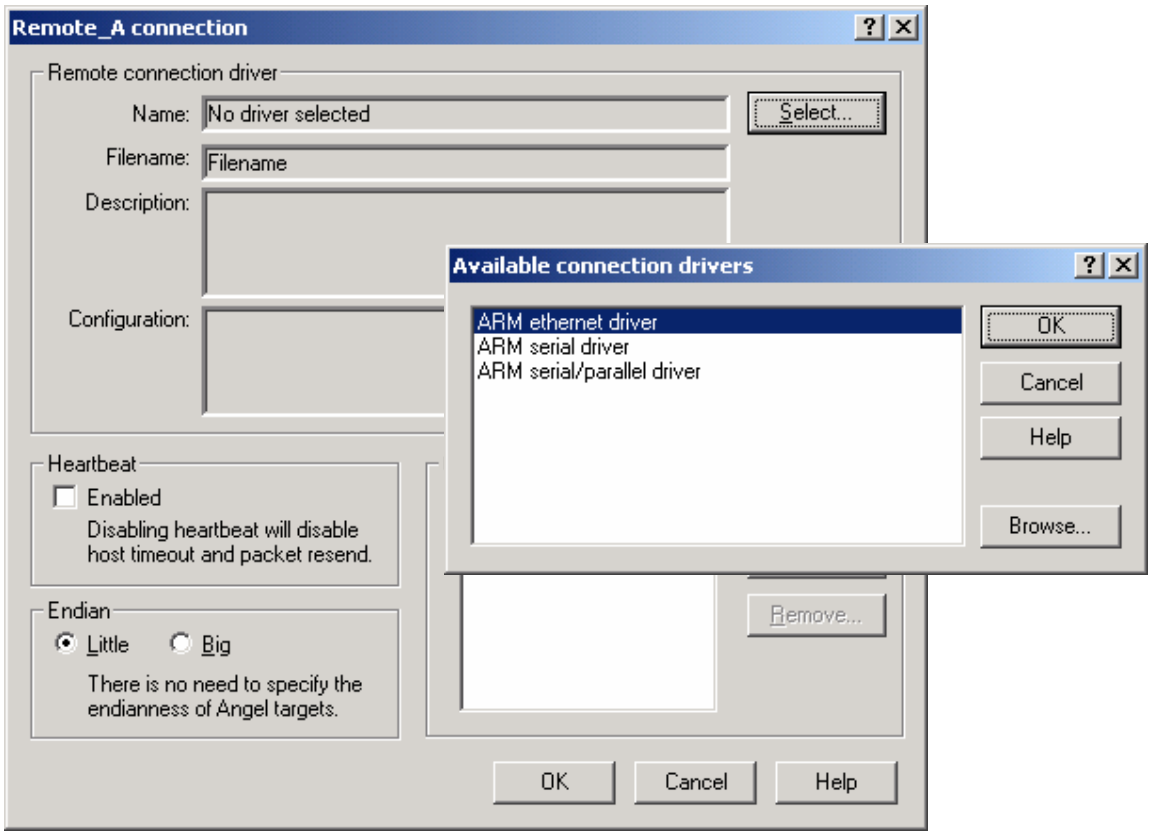
Here put some name and description if desired or leave as suggested and click OK.


Create New RDI Target
Enter a name and a description for the new entry in the connection list:
Short Name (example - "Dual 7TDMI"):
Remote_A.dll
Description (example - "Multi-ICE with two ARM7s"):
C:\Program Files\ARM\RDI\Targets\Remote_A\1.3.1\9
OK
Cancel

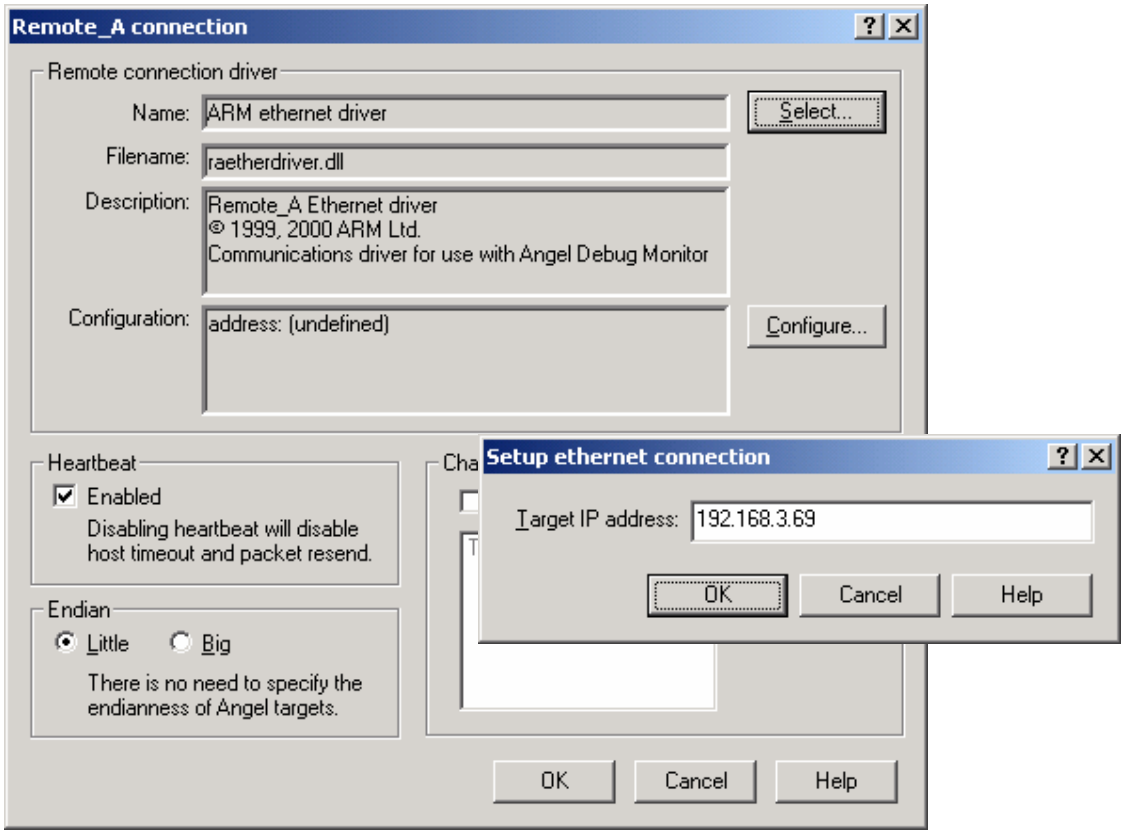
Now select the new created item in the RDI Target List and click Configure:



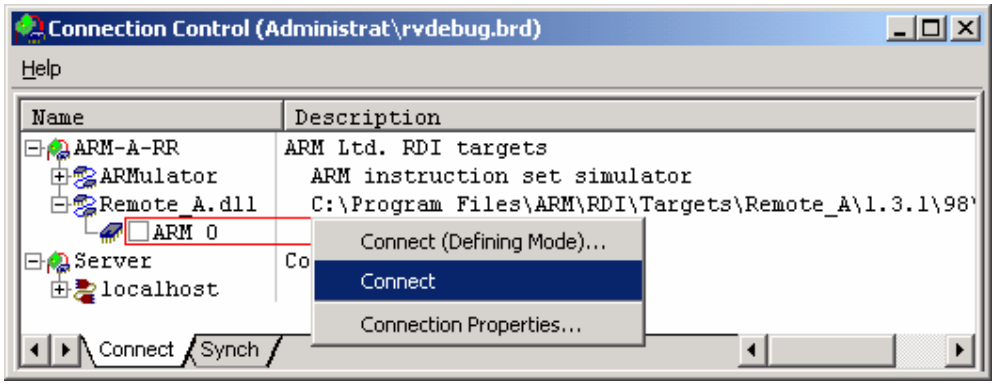
In the dialog click Select, and select ARM Ethernet driver.



Than click Configure and enter the IP address of your PEEDI and click OK on both dialogs to close them:



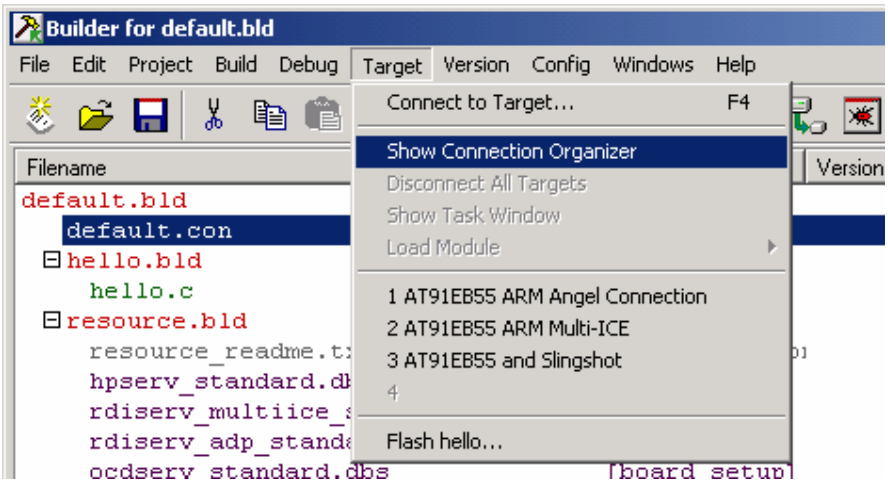
Here expand the new created item and right-click on its sub-item. On the appeared context menu click Connect to connect to the target.



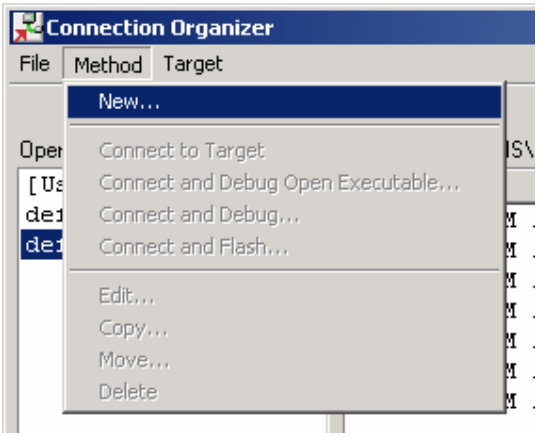
At this point you should be able to load and debug the project executable image.

3.11.3 Green Hills MULTI v3.6 for ARM

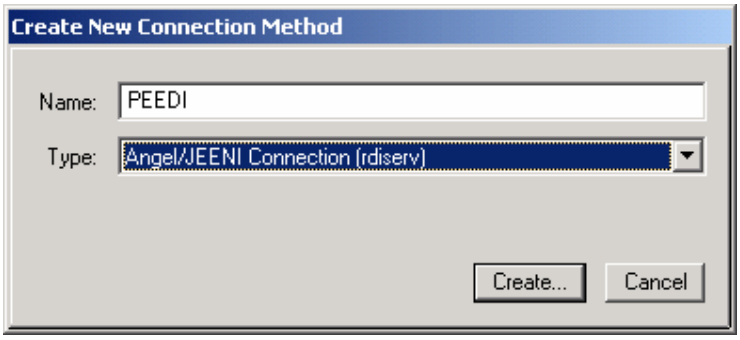
Click Target->Show Connection Organizer to open the Connection Organizer:



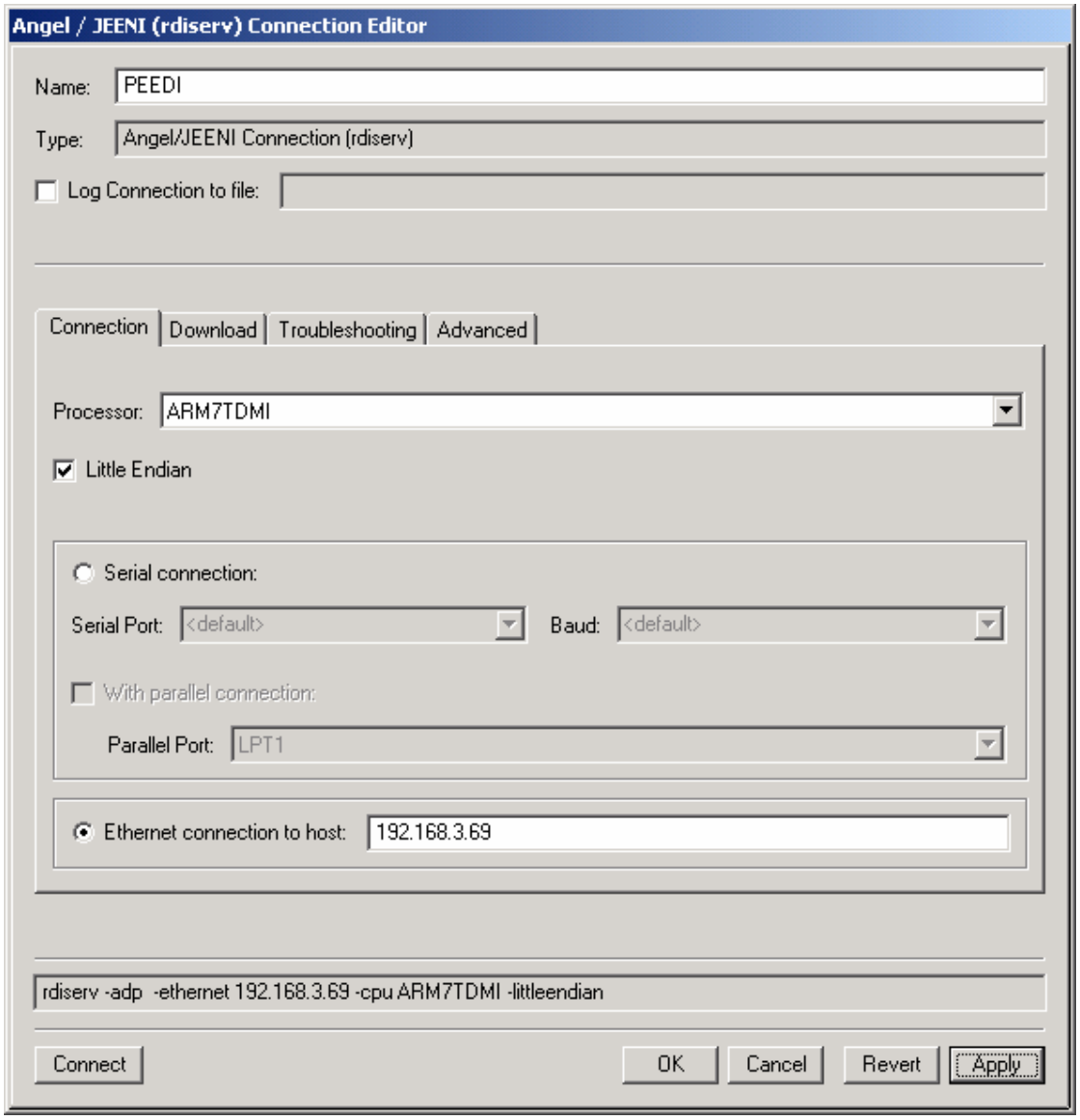
In the Connection Organizer click Method->New… to open Create New Connection Method dialog:



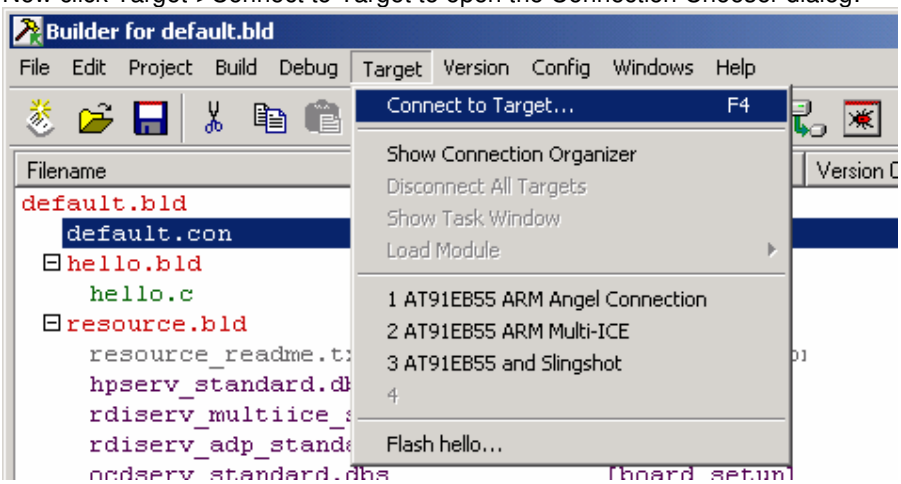
In Create New Connection Method dialog type some connection name, select Angel/JEENI Connection (rdiserv) type and click Create…:



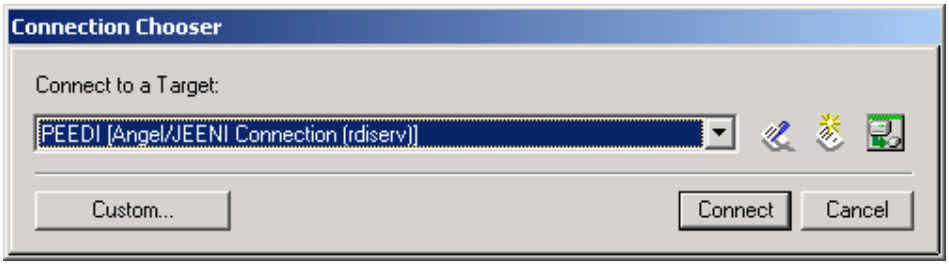
In the just opened Connection Editor select Processor and other options you desire, the important thing here is to select Ethernet connection and fill the IP address of your PEEDI. Than click OK and close the Connection Organizer.



Now click Target->Connect to Target to open the Connection Chooser dialog:



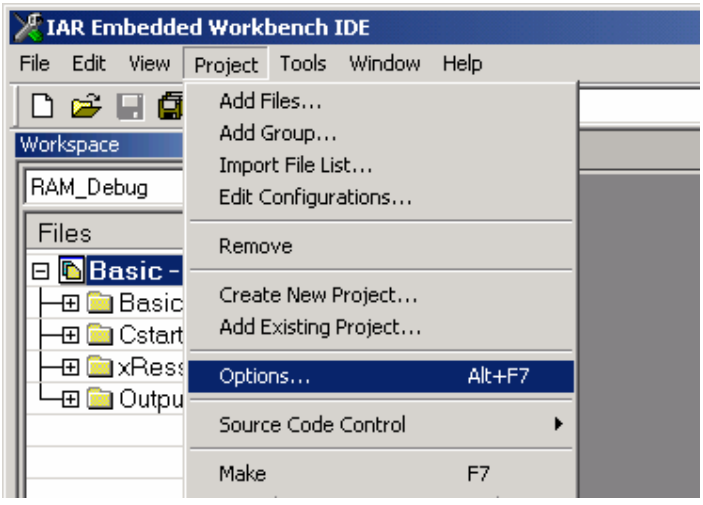
In the Connection Chooser dialog select the new created connection and click Connect:



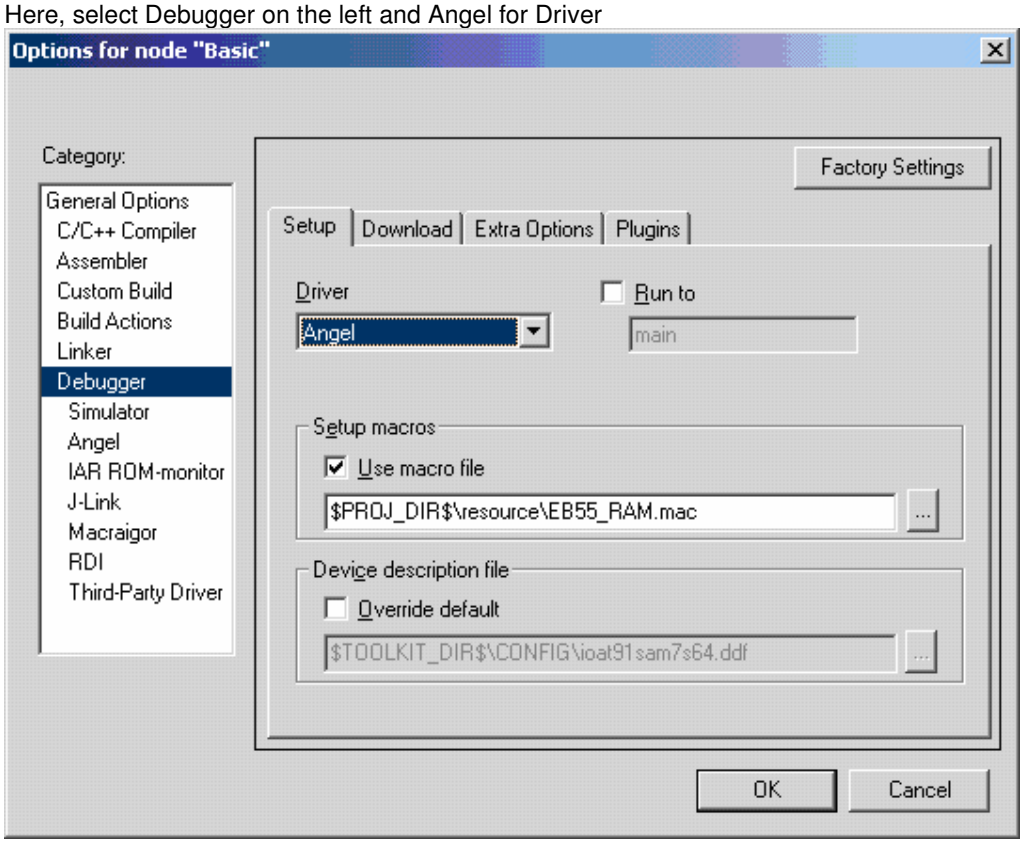
Now you should be connected to PEEDI and may start debugging.

3.11.4 IAR Embedded Workbench for ARM v4.30

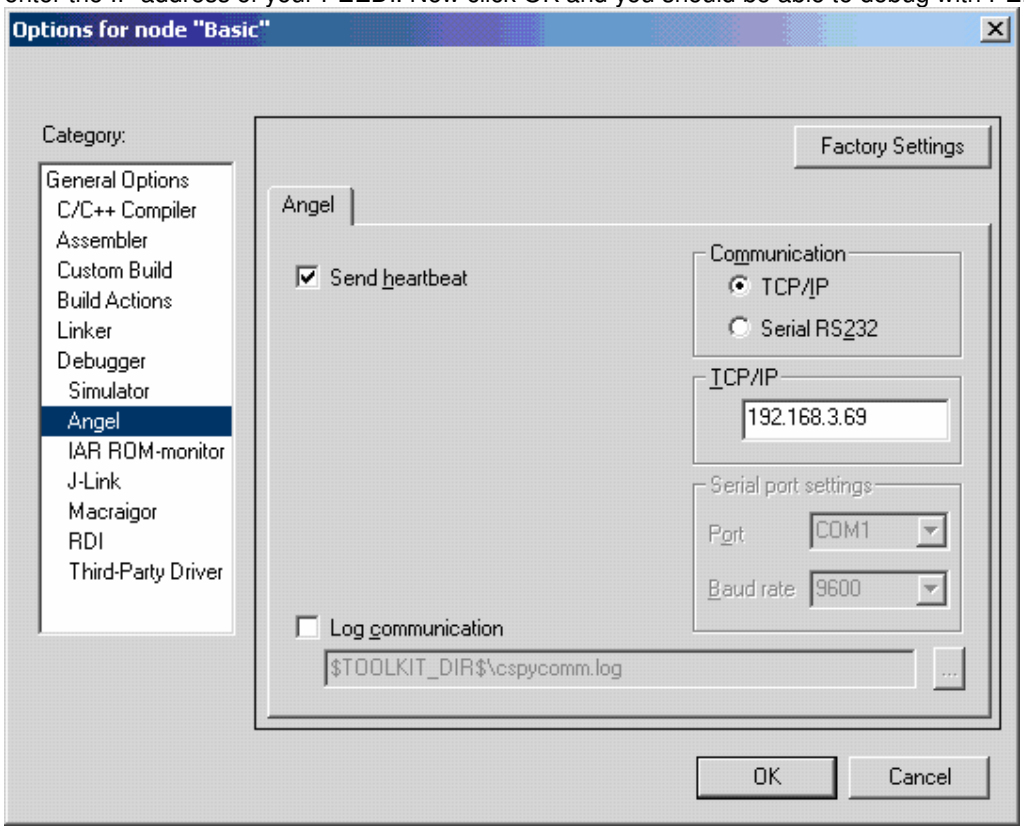
In the Embedded Workbench IDE click Project->Options… to open the options dialog:



Here, select Debugger on the left and Angel for Driver



Than click Angel on the left to show the Angel connection options, select TCP/IP for communication and enter the IP address of your PEEDI. Now click OK and you should be able to debug with PEEDI



3.12 Debugging Linux kernel

To debug the kernel you bootloader must be set to load and start the kernel successfully.
In the target configuration file set the COREn_STARTUP_MODE to RESET and in the INIT section add this for all targets except Xscale:

```
break add hard 0x90000398 ;addr from 'nm vmlinux | grep start_kernel'
go                        ; start CPU
wait 30000 stop           ; wait 30 seconds to enter debug
break del all             ; remove previously added watchpoint
beep 500 20               ; beep to signalize ready for debug
```

For Xscale targets use this:

```
break add watch 0xffff001C w 32 ; watch point on setting vectors
go                              ; start CPU
wait 30000 stop                 ; wait 30 seconds to enter debug
break del all                   ; remove previously added watchpoint
beep 500 20                     ; beep to signalize ready for debug
```

This will set a break/watch in the beginning of the Linux kernel code and will start the kernel. This way after target is powered the kernel will be started and a little later it will enter debug. At this point you can start gdb/insight pointing the kernel ELF image file. Next you can use the **target** command to connect to PEEDI. Make a **si** just for the gdb/insight to refresh its source window. Now you can set/remove breakpoints in you source code, step-by-step examine the execution or issue **continue** to start the kernel after you have set all the breakpoints you desire. If a break point is hit, gdb/insight will highlight the source line where the execution has stopped.

3.13 Target OS tasks awareness and debugging uClinux applications

PEEDI provides target OS tasks awareness, i.e. when debugging, the **info threads** gdb command or Insight's View->Thread List will list you all the OS tasks (processes/threads). But before using **info threads**, you must first set in the target configuration file a section that tells PEEDI how to find the tasks. This section includes addresses and offsets you need to fill in order for PEEDI to be able to scan the OS task list.

First add a COREn_OS CFG parameter and set is to point to the OS section like this:

```
CORE_OS = OS_ARM_LINUX_v26              ; section which contains the OS parameters
```

For Linux the section should look like this:

```
[OS_ARM_LINUX_v26]
BASE          = 4, 0xC02FF068     ; address of init_task structure
NEXT          = 4, -0xB4          ; offset of the field that points to the next item
PID           = 4, 0xE0           ; offset of the field containing the task PID
NAME          = 16, 0x1FC         ; offset of the field containing the task name

CONTEXT       = 10*4, 0x4, 0x1C   ; offset of the field containing the task context
CONTEXT_REG   = r4, 0x00          ; list of reg names and their offsets in the context
CONTEXT_REG   = r5, 0x04
CONTEXT_REG   = r6, 0x08
CONTEXT_REG   = r7, 0x0C
CONTEXT_REG   = r8, 0x10
CONTEXT_REG   = r9, 0x14
CONTEXT_REG   = r10,0x18
CONTEXT_REG   = r11,0x1C
CONTEXT_REG   = sp, 0x20
CONTEXT_REG   = pc, 0x24
```

For Linux the section should look like this:

```
[OS_ARM_UCLINUX_v26]
```

```
BASE          = 4, 0xC02FF068     ; address of init_task structure
NEXT          = 4, -0xB4          ; offset of the field that points to the next item
PID           = 4, 0xE0           ; offset of the field containing the task PID
NAME          = 16, 0x1FC         ; offset of the field containing the task name

CODE_START    = 4, 0xBC, 0x48     ; offset of the field containing the task code addr.
DATA_START    = 4, 0xBC, 0x48+8   ; offset of the field containing the task data addr.
BSS_START     = 4, 0xBC, 0x48+12  ; offset of the field containing the task data addr.

CONTEXT       = 10*4, 0x4, 0x1C   ; offset of the field containing the task context
CONTEXT_REG   = r4, 0x00          ; list of reg names and their offsets in the context
CONTEXT_REG   = r5, 0x04
CONTEXT_REG   = r6, 0x08
CONTEXT_REG   = r7, 0x0C
CONTEXT_REG   = r8, 0x10
CONTEXT_REG   = r9, 0x14
CONTEXT_REG   = r10,0x18
CONTEXT_REG   = r11,0x1C
CONTEXT_REG   = sp, 0x20
CONTEXT_REG   = pc, 0x24
```

For eCos the section should look like this:

```
[OS_ARM_ECOS_v2]
BASE          = 4, 0x6a74, 0      ; address of init_task structure
NEXT          = 4, 0xEC           ; offset of the field that points to the next item
PID           = 2, 0x58           ; offset of the field containing the task PID
NAME          = 32, 0xE8, 0       ; offset of the field containing the task name

CONTEXT       = 19*4, 0x0C, 0     ; offset of the field containing the task context
CONTEXT_REG   = r0, 0x00          ; list of reg names and their offsets in the context
CONTEXT_REG   = r1, 0x04
CONTEXT_REG   = r2, 0x08
CONTEXT_REG   = r3, 0x0C
CONTEXT_REG   = r4, 0x10
CONTEXT_REG   = r5, 0x14
CONTEXT_REG   = r6, 0x18
CONTEXT_REG   = r7, 0x1C
CONTEXT_REG   = r8, 0x20
CONTEXT_REG   = r9, 0x24
CONTEXT_REG   = r10,0x28
CONTEXT_REG   = r11,0x2C
CONTEXT_REG   = r12,0x30
CONTEXT_REG   = sp, 0x34
CONTEXT_REG   = lr, 0x38
CONTEXT_REG   = pc, 0x48 ; the PC placeholder is not filled so we use svc_lr instead
CONTEXT_REG   = cpsr, 0x40
```

Debugging uClinux applications

Since uClinux is a MMU-less port of Linux and loads applications at different addresses every time, thus it is hard to debug applications because the debugger does not know applications' sections' addresses. To tell the gdb the addresses you need first use the **info threads** and then the **add-symbol-file** command like this:

```
(peedi) info threads
11 thread 18 (spar: 0xF64040 -s .data 0xF4A004 -s .bss 0xF53104)  0x00f6918c in ?? ()
10 thread 17 (sh: 0xF89040 -s .data 0xFA1004 -s .bss 0xFA40B4)  0x00f6918c in ?? ()
9 thread 16 (telnetd: 0xFAC040 -s .data 0xFA8004 -s .bss 0xFA9984)  0x00f6918c in ??
8 thread 15 (inetd: 0xFB8040 -s .data 0xFD9004 -s .bss 0xFDA544)  0x00f6918c in ?? ()
7 thread 7 (mtdblockd)  0x00f6918c in ?? ()
6 thread 6 (kupdated)  0x00f6918c in ?? ()
5 thread 5 (bdflush)  0x00f6918c in ?? ()
4 thread 4 (kswapd)  0x00f6918c in ?? ()
3 thread 3 (ksoftirqd_CPU0)  0x00f6918c in ?? ()
2 thread 2 (keventd)  0x00f6918c in ?? ()
1 thread 1 (init: 0xFF0040 -s .data 0xFC4004 -s .bss 0xFC4F44)  0x00f6918c in ?? ()

(peedi) add-symbol-file spar.gdb 0xF64040 -s .data 0xF4A004 -s .bss 0xF53104
add symbol table from file "spar.gdb" at
        .text_addr = 0xf64040
        .data_addr = 0xf4a004
        .bss_addr = 0xf53104
(y or n) y
Reading symbols from /home/user/spar/spar.gdb...done.
```

3.14 Working with CLI (Command Line Interface)

PEEDI CLI allows you to:

- **Perform simple debugging**

You can load executable image into target RAM, get or set target memory or registers, put break and watch points, start, step or stop the target. For more information, see the description of **core, go, breakpoint, step, halt, reset, info, memory** commands.

- **Program target flash**

Full functional FLASH programmer is available, capable of programming different image file formats. For more information, see the description of **flash** command.

- **Manage files from various sources**

While in CLI, you can copy files from and to local EEPROM and SD/MMC card or FTP, TFTP or HTTP servers. You can create, remove and rename directories and files on the MMC/SD card. For more information, see the description of **transfer, card** and **eeprom** commands.

3.14.1 File path convention

PEEDI can get files from local EEPROM and MMC/SD card or TFTP, FTP and HTTP server. It can store files on all the previous locations except HTTP server. However the download speed from HTTP and FTP servers is times faster than TFTP servers. FAT12, FAT16 and FA32 formatted MMC/SD cards are supported but there is no support for long file names, so all files should be named using the 8+3 DOS name convention or using names up to twelve characters.

This file path syntax can be used to point the desired location:

Note:
If the file path is skipped the per-core default path will be used. The default core's path is defined in the target configuration file using the COREn_PATH parameter. Full path will be used in the entire manual for clear understanding.

Note:
*If the IP address is skipped the default server IP will be used. The default server IP can be set using the **fconfig** RedBoot command. Full path will be used in the entire manual for clear understanding.*

`tftp://192.168.1.1/subdirectory/file`
TFTP server *192.168.1.1* will be requested for */subdirectory/file*

`tftp:/subdirectory/file`
TFTP default server IP will be requested for */subdirectory/file*

`ftp://user:password@192.168.1.1/subdirectory/file`
FTP server *192.168.1.1* will be requested for *subdirectory/file* from the current working directory right after the login of *user* with *password*

`ftp://user:password@192.168.1.1//subdirectory/file`
FTP server *192.168.1.1* will be requested for *subdirectory/file* from server root directory using *user* and *password* credentials to login

`ftp://192.168.1.1/subdirectory/file`
FTP server *192.168.1.1* will be requested for *subdirectory/file* using user *anonymous* and password *guest* to login

`ftp:user:password/subdirectory/file`

FTP default server IP will be requested for *subdirectory/file* from the current working directory right after the login of *user* with *password*

`ftp:user:password//subdirectory/file`

FTP default server IP will be requested for *subdirectory/file* from server root directory using *user* and *password* credentials to login

`ftp:subdirectory/file`

FTP default server IP will be requested for *subdirectory/file* from the current working directory right after the login of user *anonymous* with password *guest*

`ftp:/subdirectory/file`

FTP default server IP will be requested for *subdirectory/file* from server root directory right after the login of user *anonymous* with password *guest*

`http://192.168.1.1/subdirectory/file`

HTTP server *192.168.1.1* will be requested for */subdirectory/file*

`http:/subdirectory/file`

HTTP default server IP will be requested for */subdirectory/file*

`card://subdirectory/file`

MMC/SD card will be searched for */subdirectory/file*

`eep://file`

Local EEPROM will be searched for *file*. EEPROM file system if flat, i.e. directories are not supported. Keep in mind it has very limited storage space (tenths of kilobytes).

`file`

The default path to the *file* will be used, got from the COREn_PATH configuration parameter.

3.14.2 CLI commands

PEEDI has full functional telnet command line interface (CLI), which provides many useful commands. It has vary easy to use help system and command auto complete, so instead of "flash program" you could type only "fl pr", or you could just hit TAB to auto complete the command or subcommand. If you are unsure about some command arguments – hit TAB again and the command help will be printed so you can continue writing your command line.

An expression can also be used instead of a value in a command. The expression can include only the four main math operations: +, -, * and /. And it is interpreted without priority from left to right (e.g. **memory read 4*0x1000-32 32**).

help

Syntax:

help [COMMAND [SUBCOMMAND]]

Description:

Shows help about command or a subcommand.

Arguments:

COMMAND – command which help will be shown
COMMAND SUBCOMMAD – subcommand which help will be shown

Example:

```
help
help halt
help flash program
```

transfer

Syntax:

transfer SOURCE DESTINATION

Description:

Copy file among TFTP, FTP, HTTP, MMC/SD and EEPROM.

Arguments:

SOURCE - the source file to be copied

DESTINATION - where the file to be saved

Example:

```
transfer card://dump.bin tftp://192.168.1.1/dump.bin
```
- copy file from the mmc/sd card to a TFTP server

```
transfer dump.bin ftp://user:pass@192.168.1.1/dump.bin
```
- copy file from the EEPROM card to a FTP server

```
transfer http://192.168.1.1/dump.bin tftp://192.168.1.1/dump
```
- copy file from a HTTP server to a TFTP server

type

Syntax:

type FILE

Description:

Show content of text file.

Arguments:

FILE - text file to be shown

Example:

```
type ftp://myuser:mypass@192.168.1.1/target.cfg
```

wait

Syntax:

wait MILISECONDS [stop]

Description:

Wait specified time period or wait target to stop with a given timeout. Useful when target needs some delay while executing commands in INIT section of the target configuration or script file.

Arguments:

MILISECONDS	- period to be waited in milliseconds. Actual resolution is 10 ms

Example:

```
wait 1000
wait 5000 stop
```

core

Syntax:

```
core [#CORE]
```

Description:

Show/set current core.

Arguments:

#CORE	- core number of desired core to be current

Example:

```
core
core #1
```

clock

Syntax:

```
clock init|normal|kHz
```

Description:

Switch JTAG/BDM target clock - init, normal or any desired frequency. This is useful when the INIT section is too long and takes too much time. Using this command, you can initialize in the beginning the system clock (the PLL) and then switch to normal clock. This will allow much faster execution of the INIT section.

Arguments:

init, normal or frequency in kHz	- desired clock to be used from now on

Example:

```
clock init
clock normal
clock 3000
```

adp

Syntax:

```
adp [#CORE]
```

Description:

Show/select core for the next ADP debug session.

Arguments:

#CORE	- core number of desired core to be selected

Example:

```
adp
adp #1
```

run

Syntax:

```
run #SCRIPT_NUMBER|$SCRIPT_NAME|SCRIPT_FILE
```

Description:

Execute script from the target configuration file or a file containing CLI commands.

Arguments:

#SCRIPT_NUMBER	- number associated with a scrip defined in the configuration file
$SCRIPT_NAME	- name of a scrip defined in the configuration file
SCRIPT_FILE	- file, containing CLI commands to be executed

Example:

```
run #1
run card://myscript.cmd
```

go

Syntax:

```
go [ADDRESS|#CORE|#CORE=ADDRESS|#all]
```

Description:

Start current or specified core(s). If no address is provided the core(s) will start from its current program counter (PC) value. If no core is specified, current core will be started. If argument #all is provided, all cores will be started from their current PC values.

Arguments:

ADDRESS	- address to start from
#CORE	- core to be started
#all	- all cores will be started

Example:

```
go
go 0x100040
go #0
go #0=100040
go #0=100040 #2
go #all
```

step

Syntax:

```
step [ADDRESS|#CORE|#CORE=ADDRESS|#all]
```

Description:

Step one instruction current or specified core(s). If no address is provided the core(s) will steps from its current PC value. If no core is specified, current core will be stepped. If argument #all is provided, all cores will be stepped from their current PC values.

Arguments:

ADDRESS	- address to step from
#CORE	- core to be stepped
#all	- all cores will be stepped

Example:

```
step
step 0x100040
step #0
step #0=100040
step #0=100040 #2
step #all
```

execute

Syntax:

```
execute OPCODE
```

Description:

Force CPU to execute specified instruction. Supported in MPC5500 targets only.

Arguments:

OPCODE	- opcode of instruction to be executed

Example:

```
execute 0x7C0007A4
```

set

Syntax:

```
set [coprocessor|spr|ctrl] REGISTER VALUE
```

Description:

Set target CPU register. For more information about cp15 see **info cp15** command.

Arguments:

REGISTER	- name of register to set
VALUE	- value to set

Example:

```
set r0   0x12345678 - set general purpose register
set ice8 0x12345678 - set ICE register 8\r\n"
set dfsr 0x12345678 - ARM9: set CP15 instr. Data FSR register
                      using interpreted access
set cp15 0x51AF 0x123 - ARM9: set CP15 instr. TTB register
                      using interpreted access (bit12=1)
set cp15 0x000D 0x678 - ARM9: set CP15 Process ID register
                      using physical access (bit12=0)
set MAS0 0x1234    - PowerPC: set spr register by name
set spr 624 0x1234 - PowerPC: set spr register by number
set RAMBAR 0x0     - ColdFire: set control register by name
set ctrl 0xC05 0x0 - ColdFire: set control register by address
```

halt

Syntax:

```
halt [#CORE|#all]
```

Description:

Stop current or specified core(s). If no core is specified, current will be stopped.

Arguments:

#CORE	- core to be stop
#all	- all cores will be stopped

Example:

```
halt
halt #0
halt #all
```

reset

Syntax:

```
reset [detect|reset|run|stop [MILISECONDS]]
```

Description:

Hardware reset all core on the JTAG chain causing re-initialization of each core.
If no arguments are provided last used will be taken or the reset will be performed considering the CORE_STARTUP_MODE config parameter.
Disable/enable target reset detection.

Arguments:

reset	- reset and stop the target immediately
run	- reset and leave the target running
stop MILISECONDS	- reset and stop the target after specified time
detect 0	- disable the target reset detection
detect 1	- enable the target reset detection

Example:

```
reset
reset reset
reset run
reset stop 1000
reset detect 0
reset detect 1
```

reboot

Syntax:

reboot [redboot|watchdog]

Description:

Reboot PEEDI and reload the target configuration file and re-initialize all cores.
Reboot and enter RedBoot command line.
Enable PEEDI internal watchdog

Arguments:

None

Example:

```
reboot
reboot redboot
reboot watchdog
```

echo

Syntax:

echo TEXT

Description:

Display a line of text. Useful for printing info in scripts.

Arguments:

The text to be displayed.

Example:

```
echo Initializing SDRAM...
```

jtag

Type **jtag help** in PEEDI command line for more information

beep

Syntax:

beep FREQUENCY DURATION

Description:

Beep using given frequency and duration. Useful for signaling end of scripts execution.

Arguments:

Frequency in Hz and duration in milliseconds

Example:

```
beep 1000 500
```

jtag

Type **jtag help** in PEEDI command line for more information

target

Syntax:

Target [detach|attach]

Description:

Set PEEDI debug interface in High-Z.

Arguments:

detach/attach

Example:

```
target        - show current interface state
target detach - set interface in High-Z
target attach - set interface to normal mode
```

quit

Syntax:

quit

Description:

Quit telnet session.

Arguments:

None

Example:

```
quit
```

info

Syntax:

info SUBCOMMAND

Description:

Show information about specified topic.

Arguments:

SUBCOMMAND - subcommand specifying the needed information

Example:

```
info config
```

info flash

Syntax:

```
info flash
```

Description:

Show target FLASH configuration information.

Arguments:

None

Example:

```
info flash
```

info registers

Syntax:

```
info registers [#CORE|#all] [all]
```

Description:

Show current CPU registers' values of current, specified all cores.

Arguments:

#CORE	- core to show its registers' values
#all	- list all core registers' values
all	- list all modes registers' values

Example:

```
info registers
info registers all
info registers #0
info registers #0 all
info registers #all
info registers #all all
```

info target

Syntax:

```
info target [#CORE]
```

Description:

Show general core information.

Arguments:

None

Example:

```
info target
info target #0
```

info config

Syntax:

```
info config
```

Description:

Show JTAG configuration.

Arguments:

None

Example:

```
info config
```

info ice

Syntax:

```
info ice [REGISTER] [#CORE|#all]
```

Description:

Display ICE Breaker registers

Arguments:

Example:

```
info ice
info ice ice5
info ice 5
```

info cp15, info cp14

Syntax:

```
info cp15 [0xXXXX] [#CORE|#all]
info cp14 [0xXXXX] [#CORE|#all]
```

Description:

List current CP15 registers' values.
The ARM9 control coprocessor, cp15, provides additional registers that are used to configure and control the caches, MMU, protection system, the clocking mode and other system options.
Via JTAG, CP15 registers are accessed either direct (physical access mode) or via interpreted MCR/MRC instructions.

ARM920: Physical and Interpreted access mapping to CP15 registers

Register number for physical access mode (bit 12 = 0):

15 13	12	11 9	8	7 5	4	3 0
0 0 0	0	0 0 0	i	0 0 0	x	CRn

The bit "i" selects the instruction cache (scan chain bit 33)
The bit "x" extends access to register 15 (scan chain bit 38)

Register number for interpreted access mode (bit 12 = 1):

15 13	12	11 8	7 5	4	3 0
opc_2	1	CRm	opc_2	x	CRn

The 16bit register number is used to build the appropriate MCR/MRC instruction

CRm - Specified Coprocessor Action. Determines specific coprocessor action.
Its value is dependent on the CP15 register used.
For details, refer to CP15 specific register behavior.

CRn - Determines the destination coprocessor register.

opc_1 - Defines the coprocessor specific code. Value is c15 for CP15.
opc_2 - Determines specific coprocessor operation code. By default, set to 0.

ARM926: Physical access mapping to CP15 registers

13 11	10 8	7 4	3 0
Opc_1	Opc_2	CRn	CRm

ARM94x: Physical access mapping to CP15 registers

5	4 1	0
x	CRn	i

The bit "i" selects the instruction cache (scan chain bit 32)
The bit "x" extends access to register 6 (scan chain bit 37)

Example:

```
info cp15              - show all CP15 registers

info cp15 0x51AF       - ATM920: show inst. TTB register using
                         interpreted access (bit12=1)

info cp15 0x0109       - ARM920: show inst. cache lockdown
                         register using physical access
                         (bit12=0)

info cp15 ittb
```

info spr

Syntax:

```
info spr [NAME|NUMBER] [#CORE|#all]
```

Description:

List current SPR registers' values. PowerPC targets only.

Example:

```
info spr
info spr PID
info spr 48
```

info ctrl

Syntax:

```
info ctrl [NAME|ADDRESS]
```

Description:

List current control registers' values. ColdFire targets only.

Example:

```
info ctrl
info ctrl RAMBAR
info ctrl 0xC05
```

info breakpoint

Syntax:

```
info breakpoint [#CORE]
```

Description:

List all set break and watch points of current or a specified core.

Arguments:

#CORE - core's break and watch points to be listed

Example:

```
info breakpoint
info breakpoint #1
```

memory

Syntax:

```
memory SUBCOMMAND
```

Description:

Manage target memory. Subcommand must be provided.

Arguments:

SUBCOMMAND - subcommand specifying the memory operation

Example:

```
memory read
```

memory read

Syntax:

```
memory read[TYPE ADDRESS [COUNT]]
```

Description:

Read and show target memory contents. If no arguments are provided last used will be taken, ignoring ADDRESS and starting the listing with the next address to be listed after the previous execution of memory read. Default first used arguments are 8 32-bit values at address 0x00000000.

Arguments:

TYPE - memory access

- 8 - value is 8-bits (byte long)
- 16 - value is 16-bits (half word long)
- 32 - value is 32-bits (word long)
- 64 - value is 64-bits (double word long)
- $ - value is string

ADDRESS - where the value resides

COUNT - how many consecutive values to be listed, if not provided count 1 is assumed

Example:

```
memory read   0x1000
memory read8  0x1000
memory read16 0x1000
memory read32 0x1000
memory read64 0x1000
memory read$  0x1000
memory read   0x1000 8
memory read
```

memory write

Syntax:

```
memory write[TYPE ADDRESS VALUE [COUNT]]
```

Description:

Write target memory with specified value. If no arguments are provided last used will be taken.

Arguments:

TYPE - memory access

- 8 - value is 8-bits (byte long)
- 16 - value is 16-bits (half word long)
- 32 - value is 32-bits (word long)
- 64 - value is 64-bits (double word long)
- $ - value is string

ADDRESS - where the value is to be written

VALUE - the very value

COUNT - how many consecutive values to be written, if not provided count 1 is assumed

Example:

```
memory write    0x1000 0x5555AAAA
memory write8   0x1000 0x5A
memory write16  0x1000 0x55AA
memory write32  0x1000 0x5555AAAA
memory write64  0x1000 0x55555555AAAAAAAA
memory write$   0x1000 "hi there"
memory write    0x1000 0x5555AAAA 8
memory write
```

memory or

Syntax:

```
memory orTYPE ADDRESS MASK
```

Description:

Make logical OR with, and apply to target memory.

Arguments:

TYPE - memory access

- 8 - mask is 8-bits (byte long)
- 16 - mask is 16-bits (half word long)
- 32 - mask is 32-bits (word long)
- 64 - mask is 64-bits (double word long)

ADDRESS - where the value is to be written

MASK - mask to be used for the OR operation

Example:

```
memory or    0x1000 0x5555AAAA
memory or8   0x1000 0x5A
memory or16  0x1000 0x55AA
memory or32  0x1000 0x5555AAAA
memory or64  0x1000 0x55555555AAAAAAAA
```

memory and

Syntax:

```
memory andTYPE ADDRESS MASK
```

Description:

Make logical AND with, and apply to target memory.

Arguments:

TYPE - memory access

- 8 - mask is 8-bits (byte long)
- 16 - mask is 16-bits (half word long)
- 32 - mask is 32-bits (word long)
- 64 - mask is 64-bits (double word long)

ADDRESS - where the value is to be written

MASK - mask to be used for the AND operation

Example:

```
memory and    0x1000 0x5555AAAA
memory and8   0x1000 0x5A
memory and16  0x1000 0x55AA
memory and32  0x1000 0x5555AAAA
memory and64  0x1000 0x55555555AAAAAAAA
```

memory crc

Syntax:

```
memory crc ADDRESS LENGTH [CRC]
```

Description:

Calculate or check CRC32 on a given memory region.

Arguments:

ADDRESS - beginning of region

LENGTH - length of region

CRC - crc to check

Example:

```
memory crc 0x100000 1024
memory crc 0x100000 1024 0x1DF37A8C
```

memory load

Syntax:

```
memory load [FILE [FORMAT [OFFSET]]]
```

Description:

Load image file into target memory. If no arguments are provided last used will be taken. Default first used arguments are taken from COREn_FILE of target configuration file. While file is loaded PC will be set at start of the image or at entry point if provided by the image file.

Arguments:

FILE - the image file to be loaded

FORMAT - format of image file:

bin	- binary file
ihex	- Intel HEX format
srec	- Motorola S-record format
elf	- ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
memory load tftp://192.168.1.1/image.bin bin 0x1000
memory load tftp://192.168.1.1/image.elf elf
```

memory multi load

Syntax:

```
memory multi load #CORE0 … #COREn FILE FORMAT [OFFSET]
```

Description:

Load image file into several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from COREn_FILE of target configuration file. While file is loaded PC will be set at start of the image or at entry point if provided by the image file.

Arguments:

#CORE0 … #COREn Or #all - cores to load the file to

FILE - the image file to be loaded

FORMAT - format of image file:

bin	- binary file
ihex	- Intel HEX format
srec	- Motorola S-record format
elf	- ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
memory multi load #all tftp://192.168.1.1/image.bin bin 0x1000
memory multi load #0 #2 tftp://192.168.1.1/image.elf elf
```

memory verify

Syntax:

```
memory verify [FILE [FORMAT [OFFSET]]]
```

Description:

Verify target RAM with image file. If no arguments are provided last used with the **memory load** command will be taken.

Arguments:

FILE - the image file to be verified with

FORMAT - format of image file:

bin - binary file
ihex - Intel HEX format
srec - Motorola S-record format
elf - ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
flash verify tftp://192.168.1.1/image.elf elf
flash verify tftp://192.168.1.1/image.bin bin 0x1000
```

memory dump

Syntax:

```
memory dump ADDRESS LENGTH FILE
```

Description:

Dump target memory to a file. If no arguments are provided last used will be taken.

Arguments:

ADDRESS - beginning of memory region

LENGTH - length of memory region

FILE - file to store the image. All path except HTTP server are accepted

Example:

```
memory dump 0 1024 tftp://192.168.1.1/ram.bin
```

memory test

Syntax:

```
memory test ADDRESS LENGTH
```

Description:

Test target RAM region.

Arguments:

ADDRESS - beginning of memory region

LENGTH - length of memory region

Example:

```
memory test 0x100000 1024
```

flash

Syntax:

```
flash SUBCOMMAND
```

Description:

Manage target FLASH. Subcommand must be provided.

Arguments:

SUBCOMMAND - subcommand specifying the FLASH operation

Example:

```
flash erase
```

flash set

Syntax:

```
Flash set [FLASH]
```

Description:

Show/set current FLASH target section.

Arguments:

FLASH - FLASH section number desired to be current

Example:

```
flash set
flash set 1
```

flash blank

Syntax:

```
flash blank ADDRESS [LENGTH]
```

Description:

Check FLASH region if it is blank, i.e. filled with 0xFF. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region, default is 1, if not supplied

Example:

```
flash blank 0x400000 0x1000
```

flash erase

Syntax:

```
flash erase ADDRESS [LENGTH]
flash erase chip
```

Description:

Erase all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

“flash erase chip “ - erase using the CHIP ERASE command if supported from the flash device.

Arguments:

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region in bytes, default is 1, if not supplied

Example:

```
flash erase 0x400000 0x1000
flash erase chip
```

flash lock

Syntax:

```
flash lock ADDRESS [LENGTH]
```

Description:

If supported by FLASH, lock (protect against write/erase) all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region, default is 1, if not supplied

Example:

```
flash lock 0x400000 0x1000
```

flash unlock

Syntax:

```
flash unlock ADDRESS [LENGTH]
```

Description:

If supported by FLASH, unlock (unprotect against write/erase) all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region, default is 1, if not supplied

Example:

```
flash unlock 0x400000 0x1000
```

flash query

Syntax:

flash query ADDRESS [LENGTH]

Description:

If supported by FLASH, show the lock status of all FLASH sectors that belong or overlap to the specified region. On NAND devices, show the bad block list. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region, default is 1, if not supplied

Example:

```
flash unlock 0x400000 0x1000
```

flash program

Syntax:

flash program [FILE [FORMAT [OFFSET] [erase]]]

Description:

Program image file into target FLASH. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of the currently selected FLASH section in target configuration file.

Arguments:

FILE - the image file to be programmed

FORMAT - format of image file:

- bin - binary file
- ihex - Intel HEX format
- srec - Motorola S-record format
- elf - ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

erase - if this argument is provided, all affected FLASH sectors will be pre-erased upon programming

Example:

```
flash program tftp://192.168.1.1/image.elf elf erase
flash program tftp://192.168.1.1/image.elf elf 0x1000
flash program tftp://192.168.1.1/image.bin bin 0x1000
```

flash multi erase

Syntax:

flash multi erase #CORE0 … #COREn [ADDRESS LENGTH|chip]

Description:

Erase all FLASH sectors that belong or overlap to the specified region on into several targets simultaneously. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

#CORE0
…
#COREn
Or #all - cores to erase

ADDRESS - beginning of FLASH region

LENGTH - length of FLASH region in bytes, default is 1, if not supplied

Example:

```
flash multi erase #all 0x400000 0x1000
flash multi erase #1 #2 0x400000 0x1000
```

```
flash multi erase #all chip
```

flash multi blank

Syntax:

```
flash multi blank #CORE0 … #COREn [ADDRESS LENGTH]
```

Description:

Check if blank all FLASH sectors that belong or overlap to the specified region on into several targets simultaneously. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Arguments:

#CORE0 … #COREn Or #all	- cores to check
ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region in bytes, default is 1, if not supplied

Example:

```
flash multi blank #all 0x400000 0x1000
flash multi blank #1 #2 0x400000 0x1000
```

flash multi program

Syntax:

```
flash multi program #CORE0 … #COREn FILE FORMAT [OFFSET]
```

Description:

Program image file into several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of the currently selected FLASH section in target configuration file.

Arguments:

#CORE0 … #COREn Or #all	- cores to program
FILE	- the image file to be programmed
FORMAT	- format of image file: bin - binary file ihex - Intel HEX format srec - Motorola S-record format elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset.

Example:

```
flash multi program #0 #2 tftp://192.168.1.1/image.elf elf
flash multi program  #all ftp://192.168.1.1/imge.bin bin 0x100
```

flash multi verify

Syntax:

```
flash multi verify #CORE0 … #COREn FILE FORMAT [OFFSET]
```

Description:

Check image file onto several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of the currently selected FLASH section in target configuration file.

Arguments:

#CORE0 … #COREn Or #all	- cores to program

FILE - the image file to be programmed

FORMAT - format of image file:

bin	- binary file
ihex	- Intel HEX format
srec	- Motorola S-record format
elf	- ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset.

Example:

```
flash multi verify #all tftp://192.168.1.1/image.elf elf
flash multi verify #0 #2 ftp://192.168.1.1/image.bin bin 0x100
```

flash verify

Syntax:

```
flash verify [FILE [FORMAT [OFFSET]]]
```

Description:

Verify target FLASH with image file. If no arguments are provided last used with the **flash program** command will be taken.

Arguments:

FILE - the image file to be verified with

FORMAT - format of image file:

bin	- binary file
ihex	- Intel HEX format
srec	- Motorola S-record format
elf	- ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
flash verify tftp://192.168.1.1/image.elf elf
flash verify tftp://192.168.1.1/image.bin bin 0x1000
```

flash dump

Syntax:

```
flash dump ADDRESS LENGTH FILE
```

Description:

Dump target FLASH to a file. If no arguments are provided last used will be taken.

Arguments:

ADDRESS - beginning of memory region

LENGTH - length of memory region

FILE - file to store the image. All path except HTTP server are accepted

Example:

```
flash dump 0 1024 tftp://192.168.1.1/ram.bin
```

flash read

Syntax:

```
flash read [ADDRESS [COUNT]]
```

Description:

Read and show FLASH memory contents. Useful for NAND, SPI and DataFlash FLASH types, i.e. chips which are not visible through the CPU memory map. For NOR chips the **memory read** command may be used.

Arguments:

ADDRESS - start address

COUNT - count in bytes

Example:

```
flash read   0x1000 8
flash read
```

flash info

Syntax:

```
flash info
```

Description:

Show target FLASH configuration information.

Arguments:

None

Example:

```
flash info
```

flash find

Syntax:

```
flash find [SEARCHCRITERIA]
```

Description:

List specified or all chips in FLASH data base.

Arguments:

SEARCHCRITERA - used to filter listed output, '*' and '?' wild characters are also accepted

Example:

```
flash find
flash find AT49BV160
flash find *29F*
```

flash this

Syntax:

```
flash this SUBCOMMAND
```

Description:

The **flash this** command is used to execute FLASH specific subcommand available only for the given FLASH. See below for the available commands.

Arguments:

SUBCOMMAND - FLASH specific subcommand to be executed

Example:

```
flash this hidden enter
flash this hidden exit
flash this nvmbit 2 1
```

flash this hidden

Syntax:

```
flash this hidden enter|exit
```

Description:

Enter/exit hidden ROM mode on some FLASH devices.
Once the hidden ROM mode is entered, the **flash erase** and **flash program** commands can be used on the hidden FLASH sector.

Arguments:

enter - enter hidden ROM mode

exit - exit hidden ROM mode

Example:

```
flash this hidden enter
flash this hidden exit
```

flash this nvmbit

Syntax:

```
flash this nvmbit BIT VALUE
```

Description:

Set/clear Atmel AT91SAM7 general purpose NVM bit.

Arguments

BIT - bit number

VALUE - 0 to clear or 1 to set the specified bit

Example:

```
flash this nvmbit 2 0
flash this nvmbit 2 1
```

flash this secure

Syntax:

```
flash this secure
```

Description:

Secure AT91SAM7 CPU.

Arguments

None

Example:

```
flash this secure
```

flash this option

Syntax:

```
flash this option erase
flash this option BYTE VALUE
```

Description:

Manage ST STM32 CPU option bytes.

Arguments

BYTE - byte number 0-7

VALUE - value to be written to the option byte

Example:

```
flash this option erase
flash this option 0 0xA5
```

breakpoint

Syntax:

```
breakpoint SUBCOMMAND
```

Description:

Manage target break and watch points. Subcommand must be provided.

Arguments:

SUBCOMMAND - subcommand specifying the operation

Example:

```
breakpoint list
```

breakpoint add

Syntax:

```
breakpoint add ADDRESS
```

Description:

Set software break point. Unlimited number of software break points can be set.
If address -1 or 0xFFFFFFFF is specified, only the ARM ICE registers will be set for software breakpoints, but not actual breakpoint will be set. In this case the CPU will break (enter debug) if the breakpoint pattern is met anywhere during the code execution. Suitable to embed breaks in the source of the debugged application.

Arguments:

ADDRESS - address of the break

Example:

```
breakpoint add 0x400040
breakpoint add -1
```

breakpoint add hard

Syntax:

```
breakpoint add hard ADDRESS
```

Description:

Set hardware break point. No more than two hardware break points can be set.

Arguments:

ADDRESS - address of the break

Example:

```
breakpoint add hard 0x400040
```

breakpoint add watch

Syntax:

```
breakpoint add watch ADDRESS ACCESS TYPE
```

Description:

Set watch point. Unlimited number of watch points can beset.

Arguments:

ADDRESS - address of value to be watched for access

ACCESS - access type to break on:

- r - read access
- w - write access
- rw - any access

TYPE - type of watched value

- 8 - value is 8-bit (byte)
- 16 - value is 16-bit (half word)
- 32 - value is 32-bit (word)

Example:

```
breakpoint add watch 0x400040 32 r
```

breakpoint delete

Syntax:

```
breakpoint delete ID|all
```

Description:

Delete break or watch point.

Arguments:

ID - id number of break or watch point desired to be removed, taken using breakpoint list command

all - if provided all break and watch points will be deleted

Example:

```
breakpoint delete 7
breakpoint delete all
```

breakpoint list

Syntax:

```
breakpoint list [#CORE]
```

Description:

List all set break and watch points for the current or specified core.

Arguments:

#CORE - core's break and watch points to be listed

Example:

```
breakpoint list
breakpoint list #1
```

card

Syntax:

```
card SUBCOMMAND
```

Description:

Manage MMC/SC card files. Subcommand must be provided.

Arguments:

SUBCOMMAND – subcommand specifying the operation

Example:

```
card dir
```

card cd

Syntax:

```
card cd DIRECTORY
```

Description:

Change current directory

Arguments:

DIRECTORY - directory to make current

Example:

```
card cd mydir
```

card md

Syntax:

```
card md DIRECTORY
```

Description:

Make new directory.

Arguments:

DIRECTORY - name of the directory to be made

Example:

```
card md mynewdir
```

card rd

Syntax:

```
card rd DIRECTORY
```

Description:

Remove directory. The directory must be empty.

Arguments:

DIRECTORY - directory to be removed

Example:

```
card rd mydir
```

card dir

Syntax:

```
card dir [SEARCHCRITERIA|DIRECTORY]
```

Description:

Displays a list of files and subdirectories in a directory.

Arguments:

SEARCHCRITERIA - string to filter printed output

DIRECTORY - directory which content to be listed

Example:

```
card dir
card dir *.bin
card dir mydir
```

card copy

Syntax:

card copy SOURCE DESTINATION

Description:

Copy file.

Arguments:

SOURCE	- the source file to be copied
DESTINATION	- file to be saved

Example:

```
card copy image.bin mydir/backup.bin
```

card type

Syntax:

card type FILE

Description:

Show content of text file.

Arguments:

FILE	- text file to be shown

Example:

```
card type target.cfg
```

card delete

Syntax:

card delete FILE

Description:

Delete file.

Arguments:

FILE	- file to be deleted

Example:

```
card delete target.cfg
```

card rename

Syntax:

card rename FILE NEWNAME

Description:

Rename file.

Arguments:

FILE	- file to be renamed
NEWNAME	- new file name

Example:

```
card rename image.bin backup.bin
```

eeprom

Syntax:

eeprom SUBCOMMAND

Description:

Manage EEPROM files. EEPROM file system if flat, i.e. directories are not supported. Keep in mind it has very limited storage space (tenths of kilobytes). Subcommand must be provided.

Arguments:

SUBCOMMAND	- subcommand specifying the operation

Example:

```
eeprom dir
```

eeprom dir

Syntax:

eeprom dir [SEARCHCRITERIA]

Description:

Displays a list of files

Arguments:

SEARCHCRITERIA - string to filter printed output

Example:

```
eeprom dir
eeprom dir *.txt
```

eeprom copy

Syntax:

eeprom copy SOURCE DESTINATION

Description:

Copy file.

Arguments:

SOURCE - the source file to be copied

DESTINATION - file to be saved

Example:

```
eeprom copy target.cfg backup.cfg
```

eeprom type

Syntax:

eeprom type FILE

Description:

Show content of text file.

Arguments:

FILE - text file to be shown

Example:

```
eeprom type target.cfg
```

eeprom delete

Syntax:

eeprom delete FILE

Description:

Delete file.

Arguments:

FILE - file to be deleted

Example:

```
eeprom delete target.cfg
```

eeprom rename

Syntax:

eeprom rename FILE NEWNAME

Description:

Rename file.

Arguments:

FILE - file to be renamed

NEWNAME - new file name

Example:

```
eeprom rename image.bin backup.bin
```

eeprom format

Syntax:

eeprom format

Description:

Format EEPROM file system erasing all files.

Arguments:

None

Example:

```
eeprom format
```

eeprom alias

Syntax:

```
eeprom alias [ALIAS [MEANING]]
```

Description:

List or (un)define an alias.

Arguments:

ALIAS - alias to be (un)defined

MEANING - alias meaning to be defined

Example:

```
eeprom alias
eeprom alias cl “card dir”
eeprom alias cl “”
```

3.14.3 Using aliases

Aliases are very helpful and time saving when frequently using long commands with many arguments. For example we can define an alias named **fp** for :

```
peedi> flash program tftp://192.168.1.1/dir/image.bin bin 0x100
```

and instead of writing the whole command with all its arguments we could only write:

```
peedi>%{fp}
```

Characters %{} tell the command processor that an alias is closed between the brackets. Aliases are defined using **eeprom alias** command like this:

```
peedi> eeprom alias fp “flash program tftp://192.168.1.1/dir/image.bin bin” 0x100”
```

Next time **%{fp}** is met, all defined aliases will be searched for **fp**, and the string will be interpreted as the defined meaning of the alias. To un-define an alias you can:

```
peedi> eeprom alias fp “”
```

You could use an alias with combination of an argument like this:

```
peedi> %{fp} erase
```

Or define an argument or part of it as an alias:

```
peedi> eeprom alias myserver “192.168.1.1”
```

and use it like this:

```
peedi> flash program tftp://%{myserver}/mydir/myimage.bin bin 0x100
```

or

```
peedi> eeprom alias myfile “tftp://192.168.1.1/mydir/myimage.bin”
```

and use it like this:

```
peedi> flash program %{myfile} bin 0x100
```

Or even you may define an alias using as base another alias because alias searching is recursive:

```
peedi> eeprom alias fpe “%{fp} erase”
```

3.14.4 Using scripts

Scripts are useful when series of commands are frequently executed. For example loading and executing image on the target:

```
peedi> memory load tftp://192.168.1.1/myimage.bin bin 0x20
peedi> set cpsr 0xD3
peedi> set sp 0x200
peedi> breakpoint add 0x8
peedi> go
```

Instead of typing all the commands you could create text file and put all command to be executed:

```
; MyScript.txt
memory load tftp://192.168.1.1/myimage.bin bin 0x20 ; load image
set cpsr 0xD3          ; disable interrupts
set sp 0x200           ; set stack
breakpoint add 0x8     ; set breakpoint
go                     ; start execution
```

Comments could be used in script files; command processor ignores everything after ‘;’ character to the end of the line.
If some command returns error code script execution is interrupted and error message is issued. Aliases can also be used in scripts.
Once the script file is ready it can be ran with **run** command:

```
peedi> run tftp://192.168.1.1/mydir/MyScript.txt
```

The script file could reside anywhere PEEDI could load a file – local EEPROM, MMC/SD card, TFTP, FTP, HTTP servers.

3.15 Working with the FLASH programmer

PEEDI has built-in universal FLASH programmer. The programmer is used through the **flash** CLI commands.

The programmer can program FLASH chips in two ways:
1. Directly - completely non intrusive, no target memory is used, but very slow.
2. Using small agent program which is downloaded to the target RAM (1KB) and uses configurable data buffer (0.5-64KB).

So you can choose which is best suitable for your needs, but keep in mind that the "agent" method is much faster. The programming method is set in the FLASH section of the target configuration file, where the configurations are available – DIRECT, AGENT and AUTO – where first agent is tried, if failed the direct method is used.

The image to program is not buffered to the PEEDI's RAM, but it is downloaded from a TFTP/FTP/HTTP server or a MMC/SD card and programmed in configurable data blocks (0.5-64KB), which means that there is no theoretical maximum size limit of the image to be programmed.

Using the programmer you can:
- program the FLASH chip
- verify the FLASH chip
- erase part or entire FLASH chip
- blank check the FLASH chip
- lock/unlock (if supported)

To erase the FLASH, type:

```
peedi> flash erase
```

this will erase the whole FLASH, to erase all sectors within specified FLASH region, type:

```
peedi> flash erase 0x200000 0x1000
```

To program the FLASH using the default arguments from the target configuration file, type:

```
peedi> flash program
```

To program the FLASH using specific file type in a given format to an exact address issue:

```
peedi> flash program tftp://192.168.1.1/mydir/myimage.bin bin 0x100
```

The address to program the image at must be aligned to the FLASH access width, i.e. if the FLASH is 16 bits (2 bytes) accessible the address must be aligned by 2. If the FLASH is an Intel Strata the alignment must be 32 bytes. If the internal FLASH of an Atmel AT91SAM7 series microcontroller is programmed, the alignment must be equal to the FLASH page size (128 or 256 bytes). If the internal FLASH of a Philips LPC2000 series microcontroller is programmed, the alignment must 256 bytes.

After the flash is programmed, you can verify it by:

```
peedi> flash verify
```

or

```
peedi> flash verify tftp://192.168.1.1/mydir/myimage.bin bin 0x100
```

Note:

Most of the flash commands if executed without arguments, will take the last used arguments. If executed for first time they will take their default arguments. For more information on how to use the FLASH programmer, please see the ***flash*** *CLI commands.*

3.16 Multiple FLASH support

The PEEDI FLASH programmer supports targets with multiple FLASH chips mapped at different addresses. Every FLASH must be described in separate section in the target configuration file. If multiple FLASH chips/configurations are present on the target each chip/configurations must be described in different section (see section PLATFORM_ARM). If single FLASH chip/configuration is used the 'm' integer number may be skipped. When working with the programmer the first FLASH is selected as current by default. To work on another FLASH, use the flash set command to select it. The multiple FLASH support could also be used to describe different profiles for the same FLASH, for example with different program method type or different image file specified. This way you can easy switch to the desired profile using the flash set command

3.17 Working with a MMC/SD memory card

As mentioned before PEEDI can operate autonomously i.e. without an Ethernet and a host computer. This is achieved by storing all necessary files (target configuration, image, script and other files) into a MMC or SD memory card.

WARNING:

If PEEDI is set to get its network settings from a DHCP server and if the Ethernet cable is unplugged or there is no DHCP server on the Ethernet, it may take some minutes for PEEDI to boot. To avoid this, make sure PEEDI can reach a DHCP server or set it to use a static IP address.

PEEDI can not format a MMC/SD card. The card must be FAT file system formatted in order to use it with PEEDI. There are two ways to copy the necessary files to the memory card. First is to use a MMC/SD card reader and a PC to copy the files. The second way, when no card reader is available is to copy the needed files using the PEEDI CLI **transfer** command, for this purpose you will also need a FTP, TFTP or HTTP server to copy the files from.

```
peedi> transfer tftp://192.168.1.1/mydir/MyFile.txt card://myfile.txt
```

The **transfer** command can also be used to copy files from the memory card to any file server on the Ethernet.
Before actually copy the files, you may need to create some directories, delete old files or something else. To do this use the PEEDI CLI **card** subcommands.

3.18 JTAG cable adapters

PEEDI is packed with one 20-pin standard JTAG adapter/cable, one custom 10-pin JTAG adapter/cable and a 14-pin adapter/cable is available for free upon request.
All adapters are mounted on the PEEDI JTAG connector and next the target cable is connected to the given adapter:

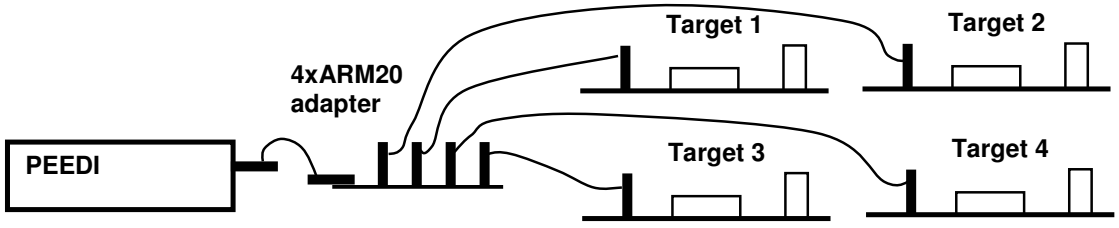
The ARM20 adapter has standard ARM pinout and may be used with almost all ARM evaluation boards.

The ARM10 adapter has no standard pinout, but it is useful when the target JTAG cable connector has to be small.

The ARM14 adapter is used for some old ARM evaluation boards.

If your target JTAG connector pinout is not standard, you may need to make your own target cable considering the PEEDI JTAG connector pinout.

The 4xARM20 adapter is used when you want to take advantage of the multiple core support. The adapter automatically shorts the unused JTAG connector pins to chain the available targets, so there is no need to set jumpers manually.



3.19 PEEDI licenses

PEEDI needs some licenses to operate. Each license unlocks specific feature of PEEDI. Licenses are kept in and loaded from the [LICENSE] section of the target configuration file. So you must fill there the licenses you have acquired before you start using PEEDI. The minimum required licenses are provided

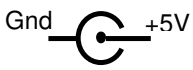
when PEEDI is purchased and are printed on the bottom side of PEEDI. Also new purchased PEEDI is set to load the target configuration file from the EEPROM, where we have put the file and the licenses.

The UPDATE_DDMMMYYYY license allows you to update PEEDI firmware to version signed to DDMMMYYYY date. This is the date when your "firmware warranty" expires (see chapter Warranty). If you update your PEEDI with firmware released after that date, your PEEDI will refuse to work. You can recover from this situation either loading older firmware or acquire a new update license, so please contact Ronetix if the UPDATE license has expired and you need to update PEEDI firmware.

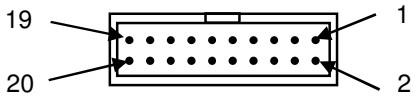
To give you a license, we need to know your PEEDI serial number, which is sent over the RS232 port when PEEDI boots or printed when you connect to PEEDI telnet CLI. The PEEDI serial number should look like this – "**SN: PD-1234-5678-90AB"**. After we receive it, we will send you the license, which should look like this – "**KEY = DESCRIPTION, 1234-5678-90AB-C"**. You have to insert that string in a new line in the [LICENSE] section of you target configuration file and reboot PEEDI.

If the license is not meant for this PEEDI, it will be simply skipped, this means that multiple PEEDIs may load single shared target configuration file, just fill in all PEEDIs' licenses.

4 Specifications

JTAG Clock	5kHz – 33MHz Adaptive Clocking
Target Voltage	1.2V – 5.0V
Network Interface	Ethernet 10/100 BaseT
Serial Interface	RS232
Power supply	5V / 1A reverse polarity protection overvoltage protection up to 100V 6.9V overvoltage shutdown
Robust Aluminum case:	
Dimensions	115x105x35mm
Weight	270 gram
LEDs:	
Power	Red
Target Power	Red
Ethernet Status	Orange
JTAG Status	Green
Buttons:	
On front panel	Two: red and green
On back panel	One: red
I/O Ports:	
JTAG Header 2x10 2.54mm pitch	Standard ESD Human Body Model IEC 1000-4-2, Direct Discharge > 4kV
RJ45	Dielectric Withstand Voltage: 1500 VAC
RS232	ESD Protection Exceeds ±15 kV Using Human-Body Model
Power Jack 2.1mm 	2.5-kV Human-Body-Model, 500-V CDM Electrostatic Discharge Protection
Operating temperature	+5ºC … +60ºC
Storage temperature	-20ºC …+80ºC
Relative humidity, non condensing	< 90%

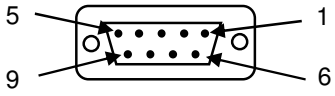
JTAG Target connector signals



Note:
Each signal JTAG pin has a 10k pull-up.

Pin	Name	Type	Description
1	**Vcc Target**	Input	**1.2V – 5.0V** Target reference voltage: used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vcc I/O on the target board.
2	**GND**		
3	**TCK**	Output	**JTAG Clock** Connects to the target TCK line
4	**GND**		
5	**TDI**	Output	**JTAG TDI** Test Data In signal from PEEDI to the target JTAG port. Connects to the target TDI line.
6	**GND**		
7	**TMS**	Output	**JTAG TMS** Connects to the target TMS line.
8	**GND**		
9	**GDBRQ**	Output	Controlled from a parameter in config file
10	**GND**		
11	**TDO**	Input	**JTAG TDO** Test Data Out signal from the target to PEEDI Connects to the target TDO line
12	**GND**		
13	**RTCK**	Input	**Returned JTAG Clock** Connects to the target RTCK line
14	**GND**		
15	**GDBACK**	Input	Not Used
16	**GND**		
17	**Reserved**		
18	**GND**		
19	**TRST**	Push-Pull or Open Drain output	**JTAG Reset** Resets the JTAG TAP controller on the target. Driver type is specified in config file
20	**RST**	Open Drain	**RESET** Resets the target system

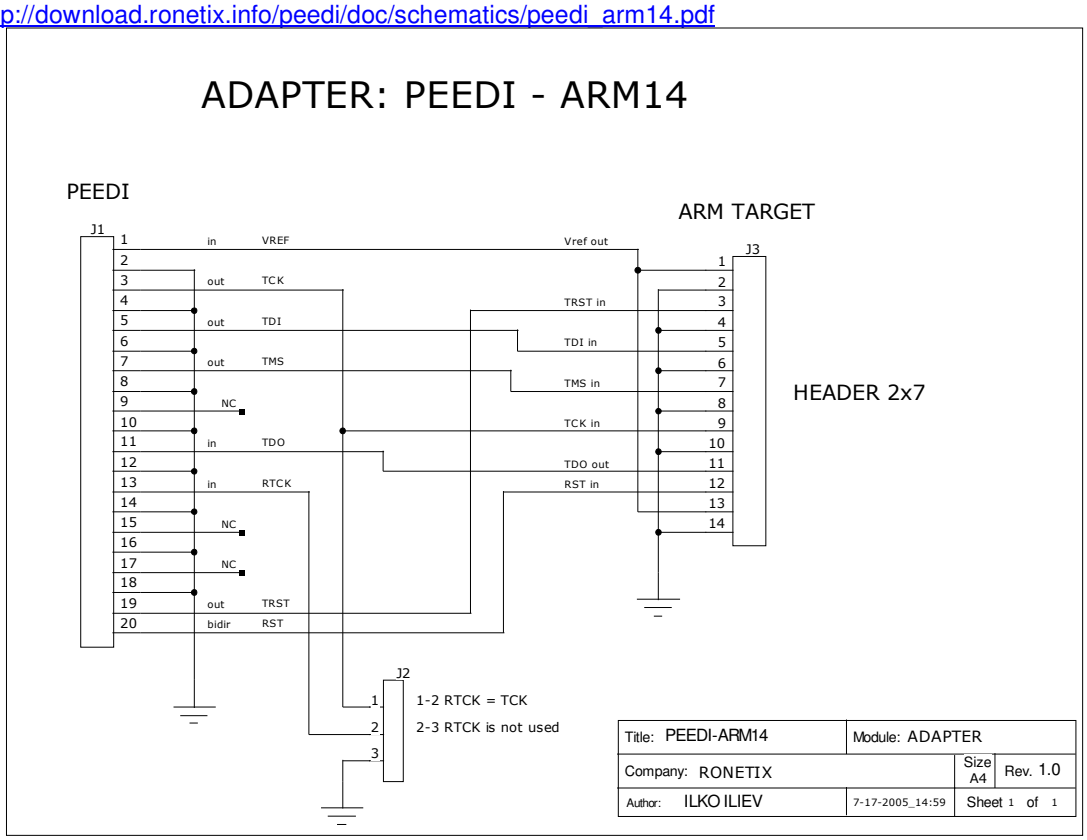
RS232 Connector (DB9F, female)



RS232 connector pin configuration	
Pin	Description
1	Not connected
2	Tx
3	Rx
4	Not connected
5	Ground
6	Not connected
7	CTS
8	RTS
9	Not connected

ARM14 JTAG cable adapter schematic

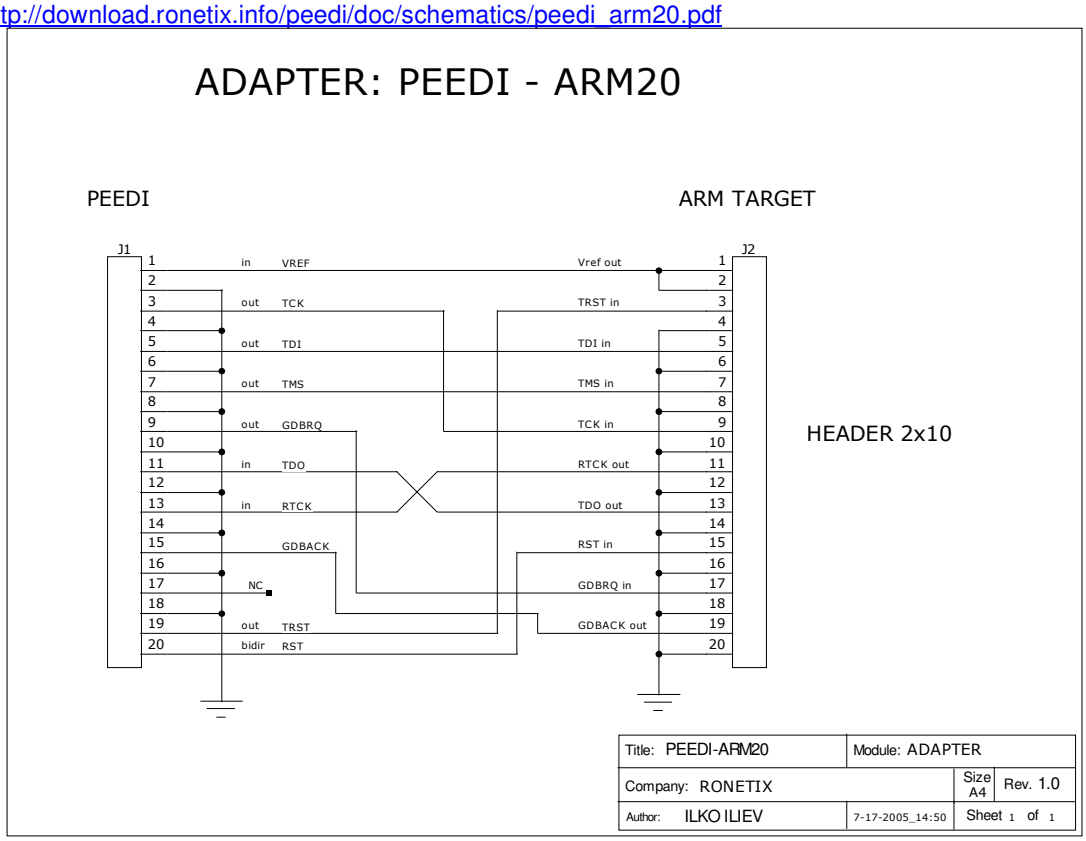
http://download.ronetix.info/peedi/doc/schematics/peedi_arm14.pdf



The ARM14 JTAG cable adepter is available for free upon request.

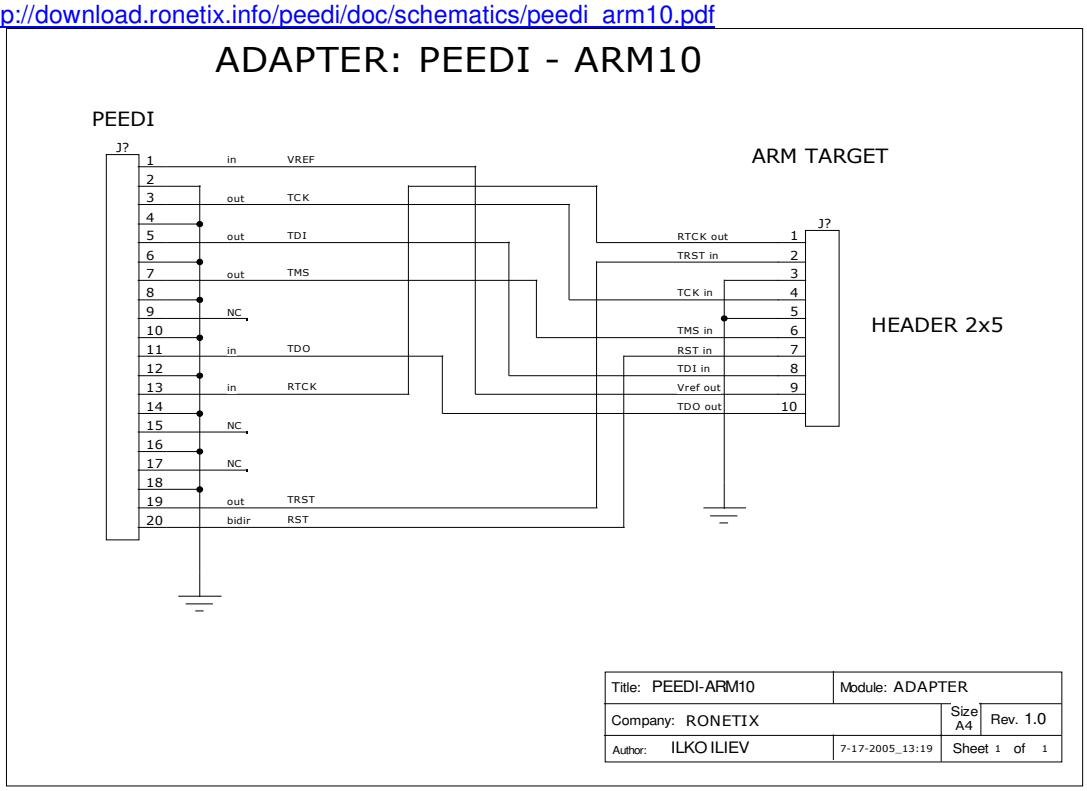
ARM20 JTAG cable adapter schematic

http://download.ronetix.info/peedi/doc/schematics/peedi_arm20.pdf



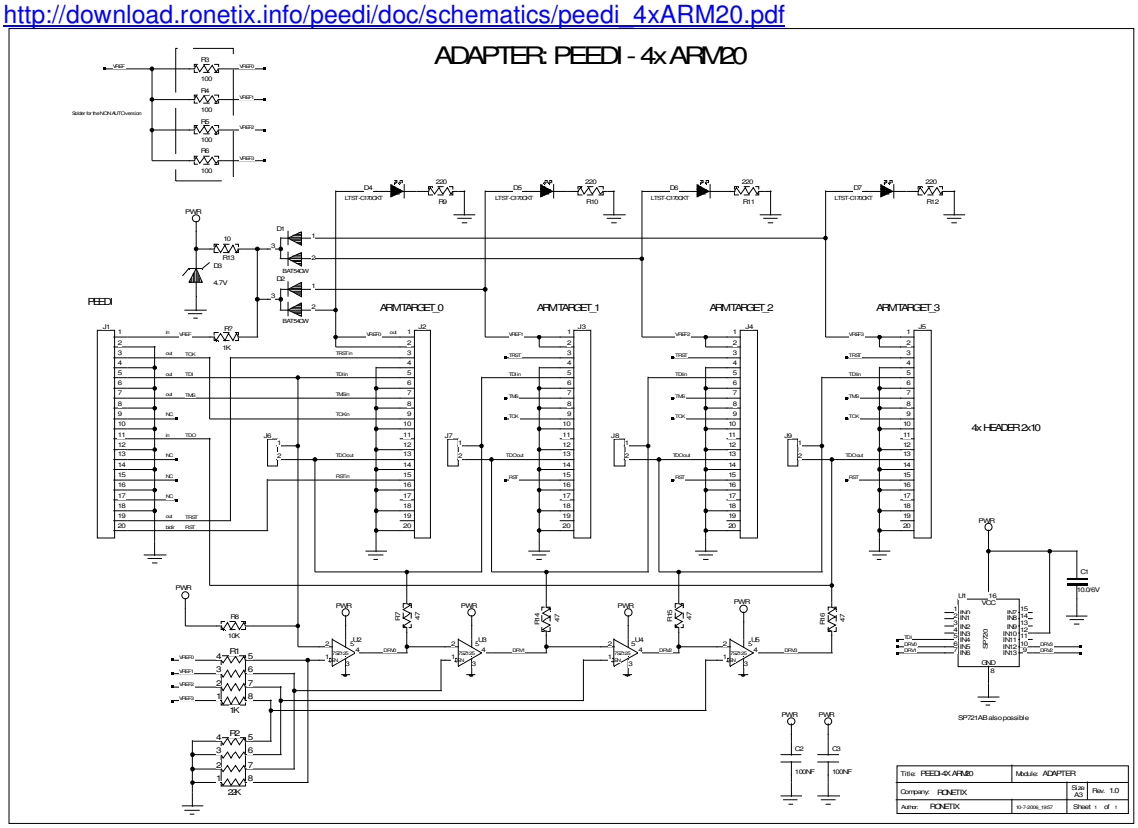
ARM10 JTAG cable adapter schematic

http://download.ronetix.info/peedi/doc/schematics/peedi_arm10.pdf



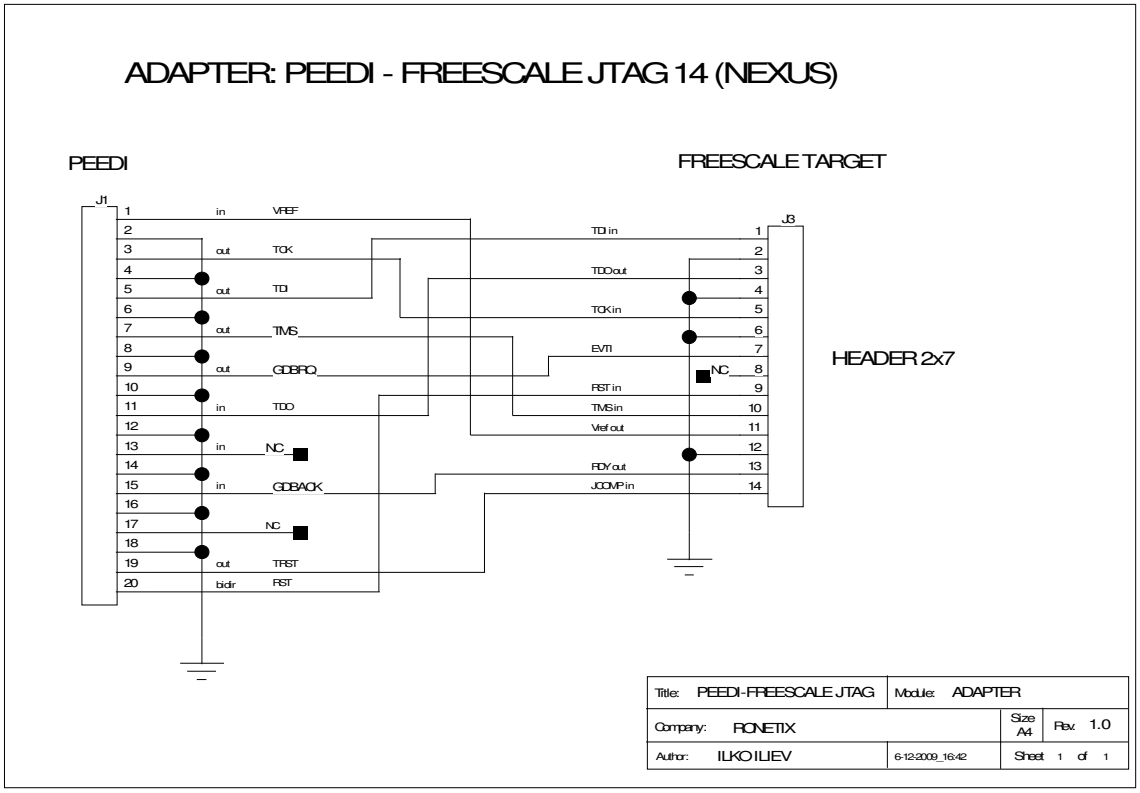
4xARM20 JTAG cable adapter schematic

http://download.ronetix.info/peedi/doc/schematics/peedi_4xARM20.pdf



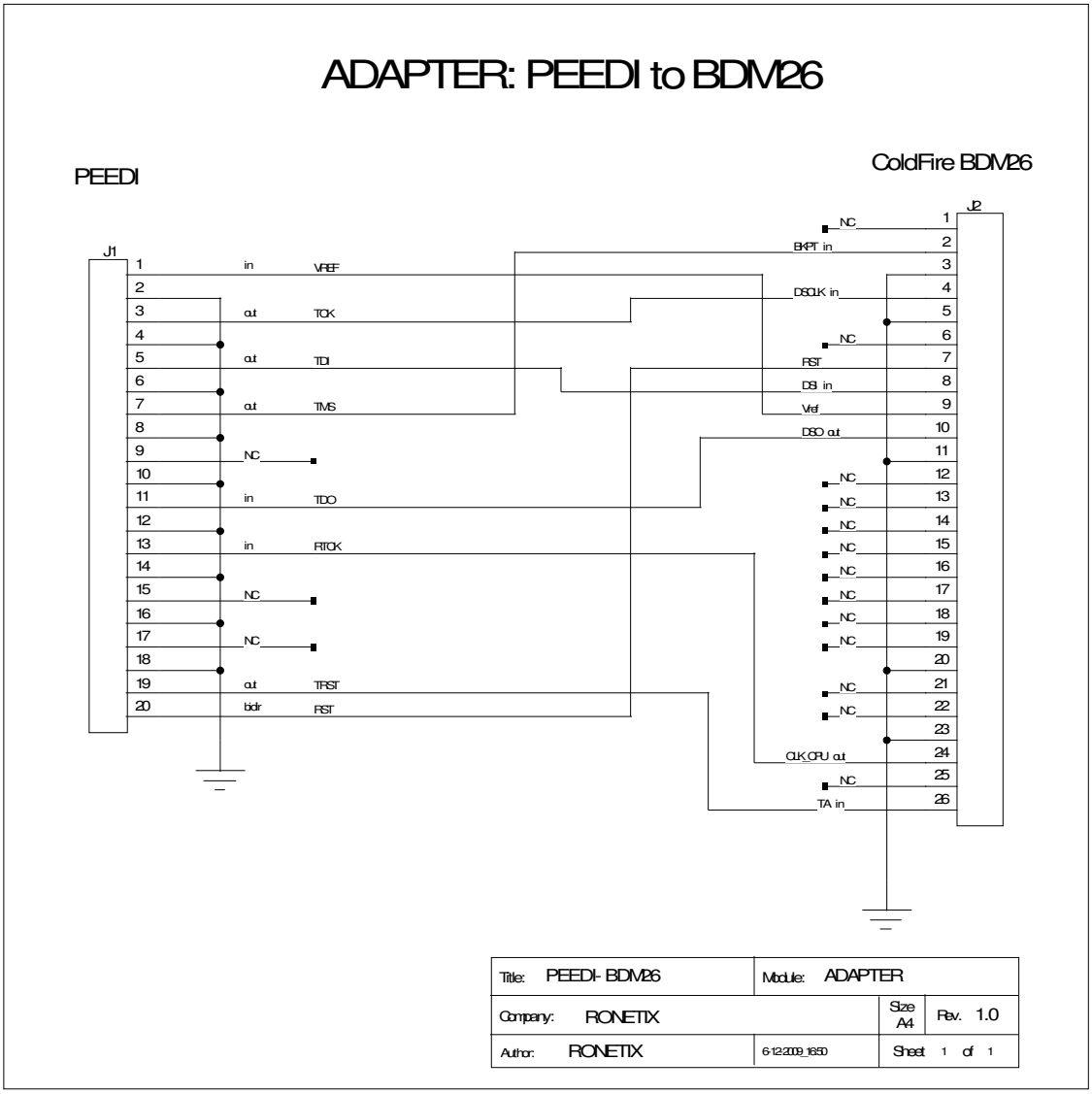
ADAPTER: PEEDI – Freescale JTAG 14

http://download.ronetix.info/peedi/doc/schematics/peedi_nexus14.pdf



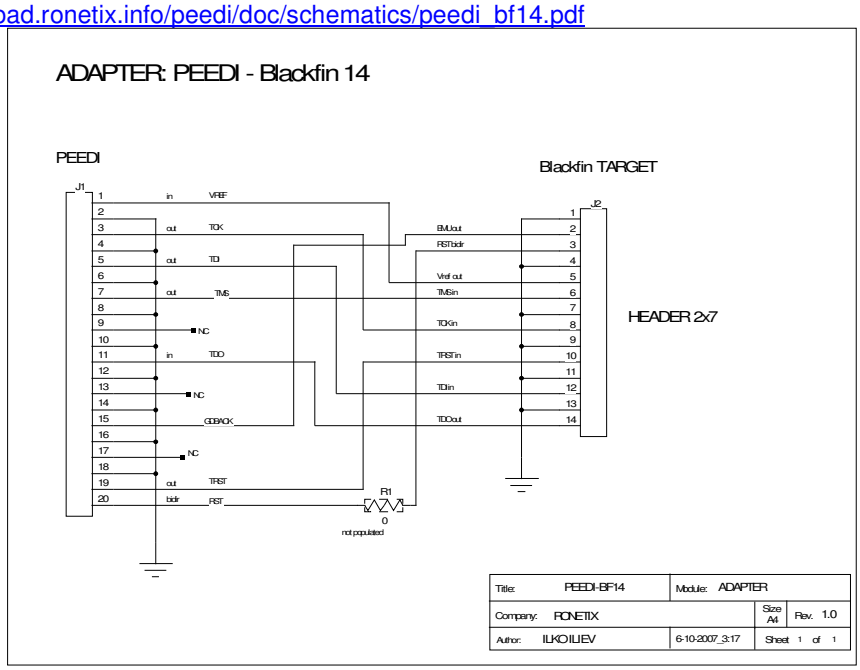
ADAPTER: PEEDI – Freescale BDM26

http://download.ronetix.info/peedi/doc/schematics/peedi_bdm26.pdf



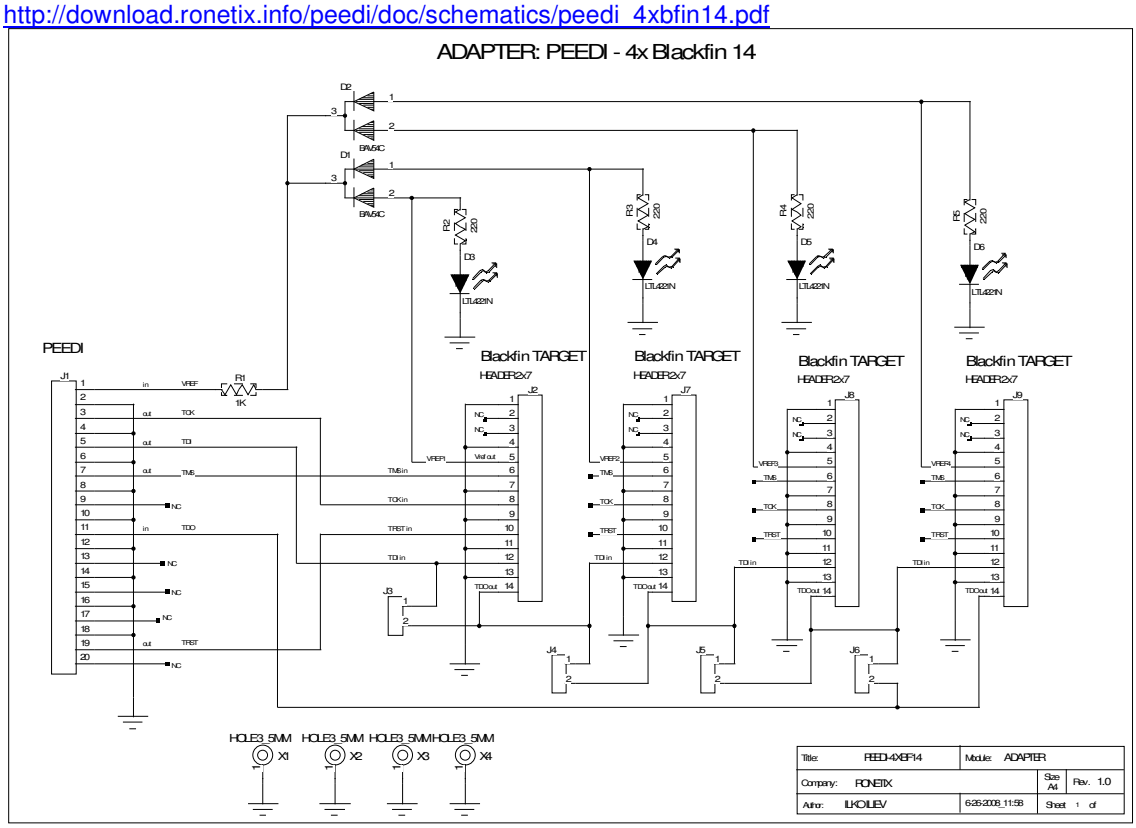
ADAPTER: PEEDI – Blackfin JTAG 14

http://download.ronetix.info/peedi/doc/schematics/peedi_bf14.pdf



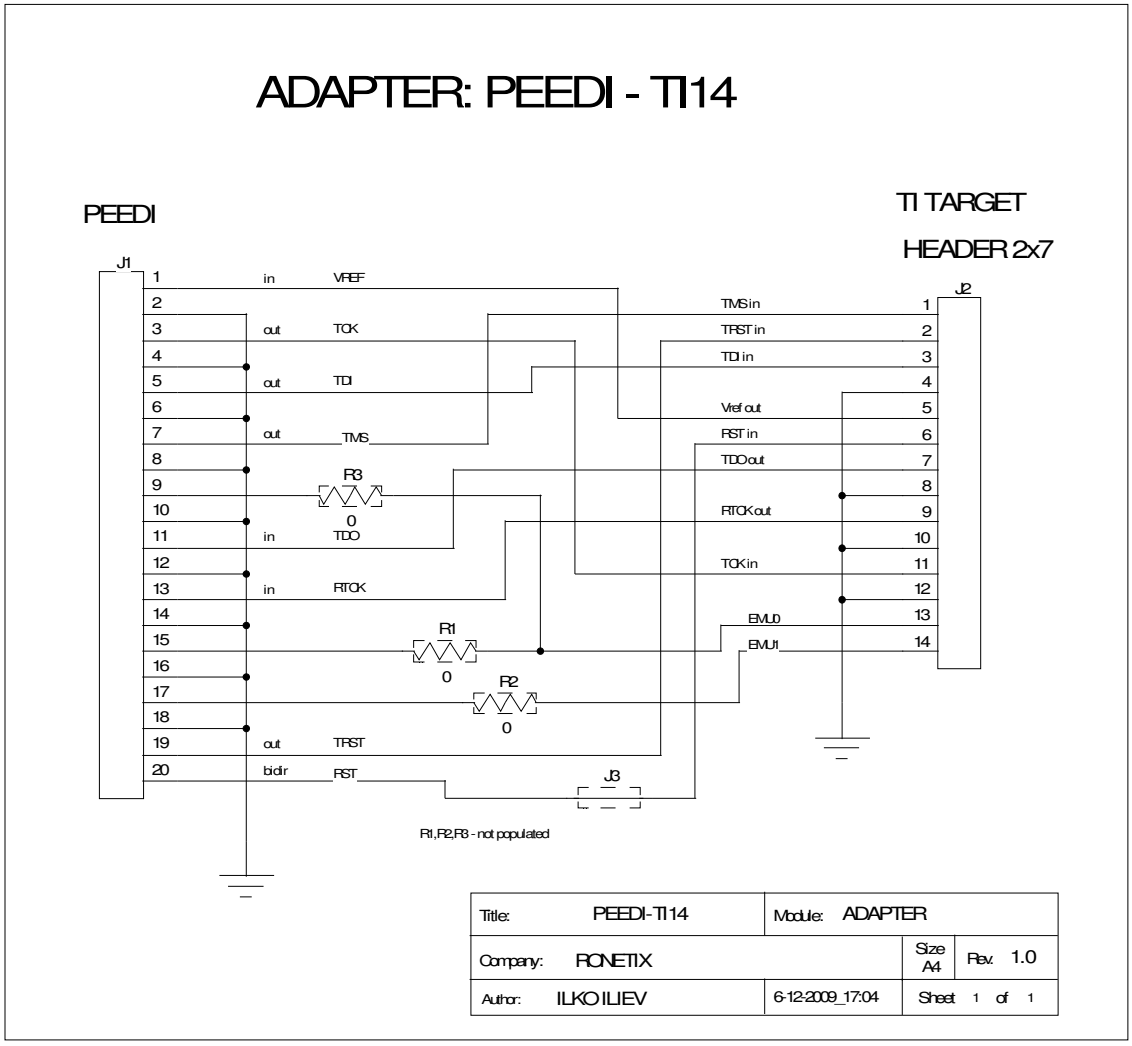
ADAPTER: PEEDI – 4x BFIN14

http://download.ronetix.info/peedi/doc/schematics/peedi_4xbfin14.pdf



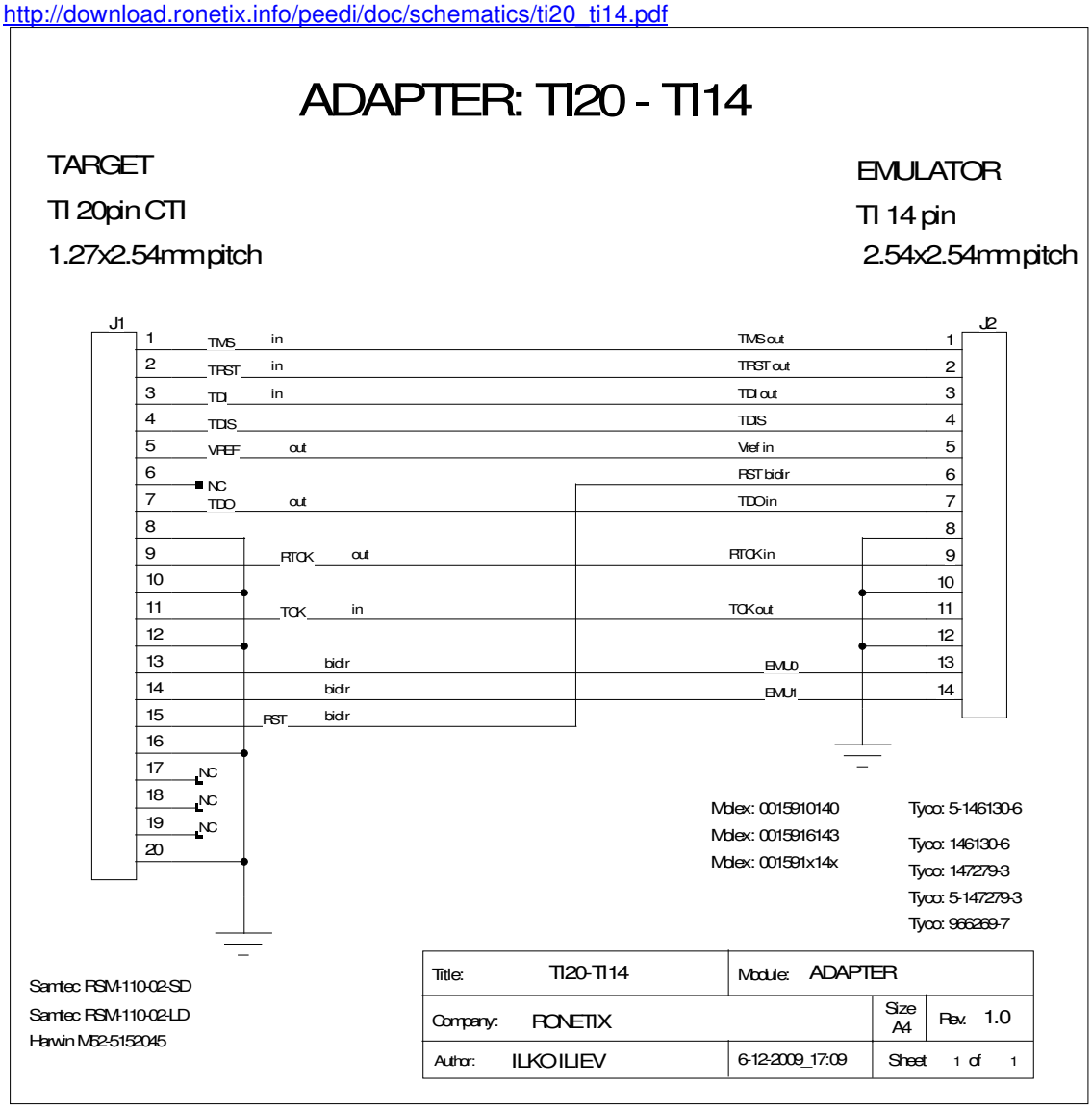
ADAPTER: PEEDI – TI JTAG 14

http://download.ronetix.info/peedi/doc/schematics/peedi_ti14.pdf



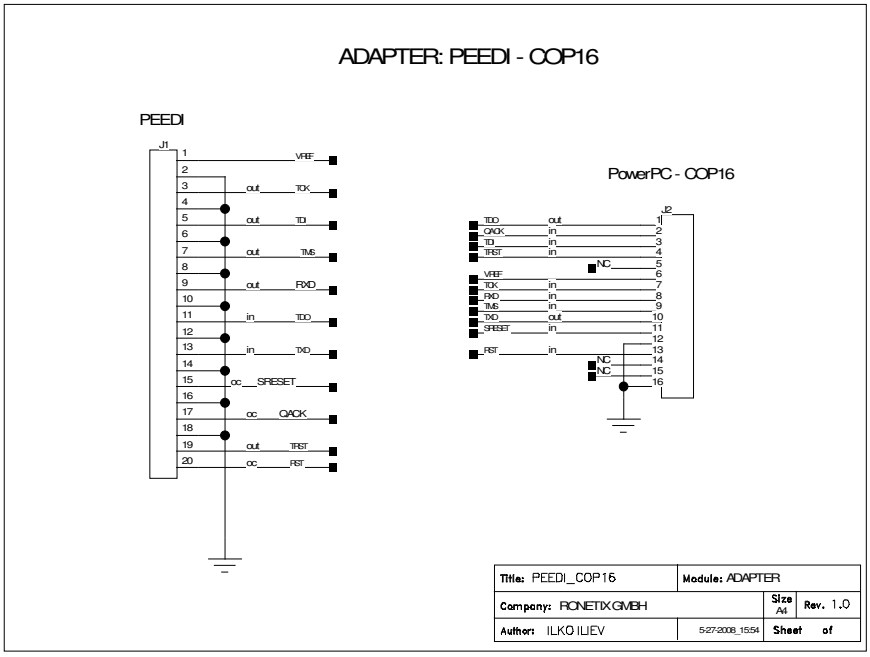
ADAPTER: TI 20 – TI14

http://download.ronetix.info/peedi/doc/schematics/ti20_ti14.pdf



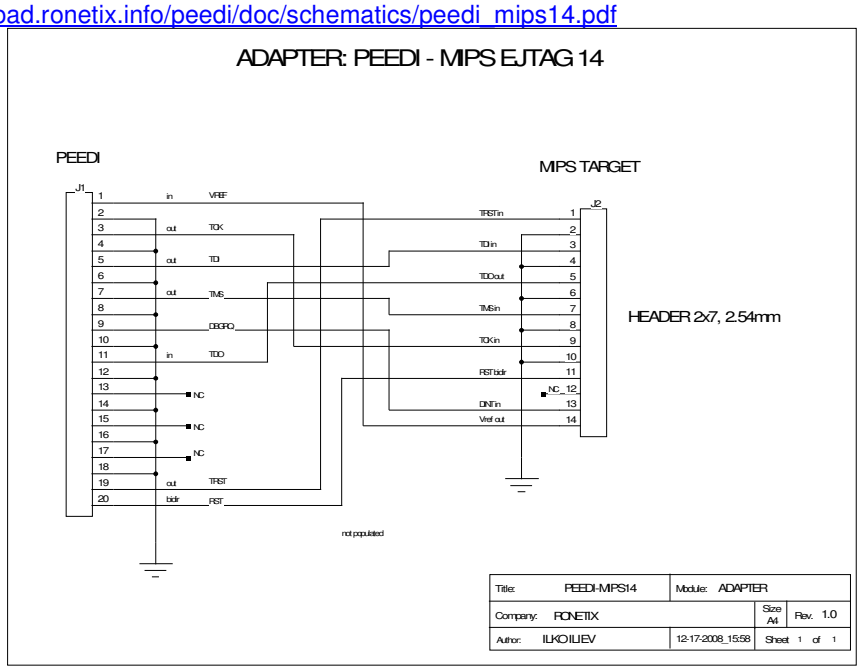
ADAPTER: PEEDI – COP16

http://download.ronetix.info/peedi/doc/schematics/peedi_cop16.pdf



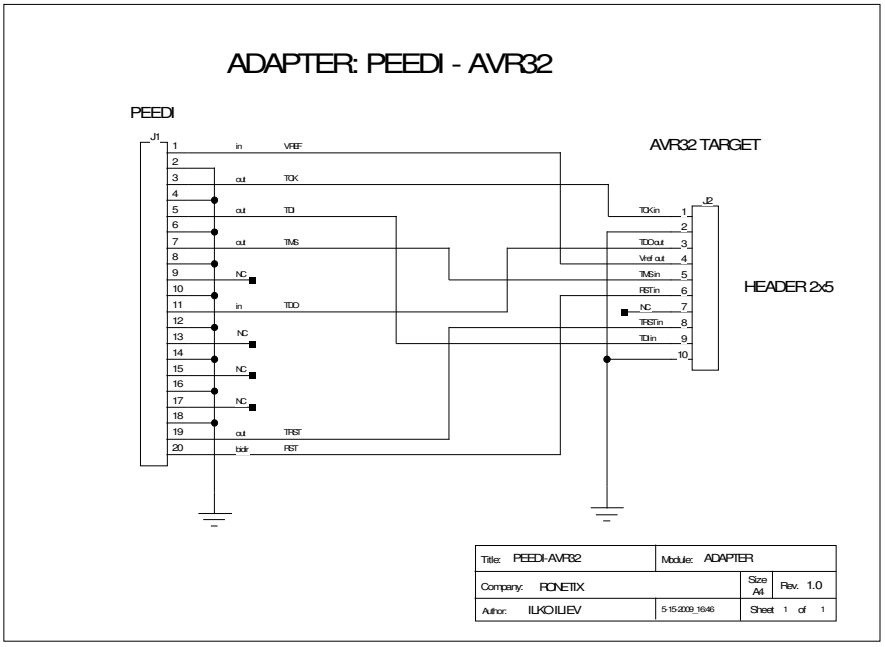
ADAPTER: PEEDI – MIPS14

http://download.ronetix.info/peedi/doc/schematics/peedi_mips14.pdf



ADAPTER: PEEDI – AVR32

http://download.ronetix.info/peedi/doc/schematics/peedi_avr32.pdf



5 FAQ

Q: What is JTAG?
A: This is a standardized high-speed serial interface, IEEE 1149, widely used for programming and debugging programmable logic and processors. It is non-intrusive, runs regardless of the state of the processor, and gives access to processor registers, memory, and other resources.

Q: What is TAP Controller?
A: The TAP controller provides access to many of the test support functions built into the JTAG-compliant device. The TAP is a state machine. The state machine controls all operations for one JTAG-compliant device. Each JTAG-compliant device has its own TAP controller. You can sequence through the state machine functions via the TCK and TMS inputs.

Q: What is EmbeddedICE?
A: EmbeddedICE is an extension to the core architecture and provides the ability to do in-circuit-emulation with deeply embedded cores.
The EmbeddedICE macrocell, adds a JTAG TAP controller and breakpoint/watchpoint logic to the ARM microcontroller which can be accessed externally through a JTAG port. Hence, software debug is facilitated by interfacing these JTAG pins of the micro to the host development system containing the ARM software development tools through a JTAG interface device such as PEEDI.

Q: What is PEEDI?
A: The PEEDI (Powerful Embedded Ethernet Debug Interface) is a debugging and development tool that provides the user the ability to see what is taking place in the target system, and control its behavior. The PEEDI probe provides the debug services that the debugger uses to perform debug operations. It receives command packets over the communication link, and translates them into the JTAG operations that are needed to provide the specific service. First, it can control the operation of the target processor and target system. What does it mean to “control” the target? In most cases it means to start and stop the processor's execution of instructions at arbitrary points in a program, examine and store values in the processor’s registers, and examine and store program code or data in the target system’s memory.

Q: What is debugging?
A: Debugging is the process of removing bugs from computer programs. On one end of the spectrum, debugging means staring at your source code until you see the bug. An infinitely more effective method is to use a special program called a "debugger".

Q: What is a debugger?
A: A debugger is a program that runs other programs. A debugger lets the user (programmer) stop running the program at any time and poke around internally. You can examine and change memory contents, call functions, and look at system registers. Besides all these fun things, a debugger can be used to fix your programs

Q: How to set gdb to work with PEEDI?
A: First compiled your application with the ‘-g –O0’ option to enable debugging. Next start gdb pointing your application:

```
$ arm-elf-insight myapp
```

To connect to the target (assuming that your PEEDI is set to use IP 192.168.1.10) type in the console window:

```
(gdb) target remote 192.168.1.10:2000
```

This will tell GDB to connect to PEEDI using remote protocol. Now you can load your application into targets memory like this:

```
(gdb) load
```

And your application is ready for debugging:

```
(gdb) continue   ; start the application
```

or

```
(gdb) si         ; make single step
```

Q: What is Eclipse?
A: The Eclipse IDE is a complete integrated development platform similar to Microsoft's Visual Studio. Originally developed by IBM, it has been donated to the Open-Source community and is now a massive world-wide Open-Source development project.

Q: What is Cygwin?
A: Cygwin is a free Linux-like environment for Windows. It works on all Windows 32-bit OS versions since Windows 95 except Windows CE. Cygwin is not a way to run native Linux apps on Windows. Applications must be rebuilt from source code to get it running on Windows.

Q: What is Cygwin/X?
A: Cygwin/X is a port of the X Window System to the Microsoft Windows family of operating systems. Cygwin/X runs on all recent consumer and business versions of Windows; as of 2003-12-27 those versions are specifically Windows 95, Windows 98, Windows Me, Windows NT 4.0, Windows 2000, Windows XP, and Windows Server 2003. For more information see http://x.cygwin.com

Q: What are GNU cross-development tools?
A: A toolchain is a collection of software tools used for the development and building of software for a particular target architecture. The GNU toolkit consists of the following software utilities:

- GCC - an ANSI C compiler
- G++ - an ANSI tracking C++ compiler
- GDB - source and assembly language command line debugger
- GAS - GNU assembler
- LD - GNU linker
- Insight - a graphical user interface for GDB

For more information see http://www.gnu.org

Q: How to enter RedBoot command line?
A: First restart PEEDI holding front panel buttons pressed, this way RedBoot will not execute its boot script and the main PEEDI application will not be loaded. Then you can access the command line via the RS232 port using suitable terminal application capable of opening the serial PC RS232 port or via telnet connecting to the port specified by the fconfig command.

Q: How to update PEEDI firmware?
A: See “Firmware update procedure”

Q: How to set target configuration file path?
A: Enter RedBoot command line and use either **fconfig** or **config** commands.
Example:

```
config new_target_cfg_file_path
```

Q: How to set the network configuration of PEEDI?
A: Enter RedBoot command line and use **fconfig** command.

Q: Why PEEDI has a display and two buttons on the front panel?
A: These are used to select, start and observe the execution of user defined scripts which contain PEEDI commands. Those scripts are defined in the target configuration file, for more information see “Using scripts”.

Q: How big image can PEEDI program?
A: The image to program is not buffered to the PEEDI's RAM, but it is downloaded from a TFTP/FTP/HTTP server or a MMC/SD card and programmed in configurable data blocks (0.5-64KB). Which means, there is no theoretical maximum size limit of the image to be programmed.

Q: PEEDI does not connect to a Philips LPC2XXX device. What should I do?
A: First make sure the pull-down resistor that enables the JTAG interface is not more than 1k and second verify that the CORE_STARTUP_MODE parameter gives the device at least 100ms to run.

Q: Whende buging mixed ARM/Thumb code using gdb/insight the debugger can not step in from ARM to Thumb function. What to do?
A: Use the **si** (step one instruction) command in the gdb/insight several times to step-in to the desired Thumb function.

6 Glossary

A

Alias – User defined alias of a command including its arguments.

Agent – Small program downloaded into the target, which is used for faster operations.

B

Breakpoint – A user-defined point where execution stops so that a debugger can examine the state of memory and registers.

Big-endian – Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word.

C

CLI – Command Line Interface.

Cygwin – Linux-like runtime environment for Windows.

Current core – The core which is set to be current, i.e. default when no core is specified.

D

Default server – Default server address used when no server is specified.

DCC – Debug Communication Channel, communication channel over the JTAG.

G

GDB – The GNU Debugger.

H

Host – A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

Hardware breakpoint – A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in Read Only Memory (ROM). Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system.

I

Insight – Graphical User Interface of GDB.

Image – An executable file that has been loaded onto a processor for execution.

J

JTAG – Type of interface which enables direct access to most CPU resources.

M

MMC/SD card – Multi Media or Secure Digital memory card, used to store files.

P

PC – Program Counter, CPU register that holds the address of the next instruction to be executed.

R

RedBoot – The Red Hat boot loader used for update, setting some configuration parameters or to load and launch the PEEDI executable image.

S

Script – List of CLI commands executed one by one until the last or until an error is returned.

T

Target configuration file – File used to describe target specifics loaded at boot-up.

7 PEEDI Package contents

Make sure all the items listed below are present, when opening the PEEDI package:

- PEEDI
- Power adapter 5V / 1A
- JTAG or BDM cable and adapter
- Patch cable CAT5, 2m
- Serial cable, 1:1, 2m
- Software CD

8 Warranty

RONETIX warrants PEEDI to be free of defects in materials and workmanship for a period of 24 months following the date of purchase when used under normal conditions.

In the event of notification within the warranty period of defects in material or workmanship, RONETIX will replace defective PEEDI. The remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including but not limited loss of profit, special, incidental, consequential, or other similar claims. RONETIX specifically disclaims all other warranties - expressed or implied, including but not limited to implied warranties of merchantability and fitness for particular purposes - with respect to defects in PEEDI, and the program license granted herein, including without limitation the operation of the program with respect to any particular application, use, or purposes. In no event shall RONETIX be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages. Failure in handling which leads to defects are not covered under this warranty. The warranty is void under any self-made repair operation except exchanging the fuse.

RONETIX warrants PEEDI firmware for a period of 12 months following the date of purchase, i.e. every reported bug will be fixed and an update will be made available.

Appendix A: Sample target configuration files

Please use this link to download the most recent version of the sample target configuration files:

http://download.ronetix.info/peedi/cfg_examples/peedi_config_files.zip